ویرایش سوم

جلد اول

یادگیری ماشین عملی با

Scikit-Learn, Keras, TensorFlow

مفاهیم، ابزارها و تکنیک‌ها
برای ساخت سیستم‌های هوشمند

پشتیبانی شده با jupyter

اُورِلیِن ژیرون

مترجم محسن زارع

© 2025 Mohsen Zare. All rights reserved.

یادگیری ماشین عملی با

Scikit-Learn ، Keras و TensorFlow

ویرایش سوم

مفاهیم، ابزارها و تکنیک‌ها برای ساخت سیستم‌های هوشمند

اُورِلیِن ژیرون

مترجم: محسن زارع

© ۲۰۲۵ محسن زارع. تمامی حقوق محفوظ است.

m7zare@gmail.com

© 2025 Mohsen Zare. All rights reserved.

یادداشت مترجم

دوستای عزیز سلام 🌷

کتابی که دستتون رسیده ترجمه‌ی فارسی ویرایش سوم کتاب معروف Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow اثر Aurélien Géron هست. این کتاب سال ۲۰۲۲ توسط انتشارات معتبر O'Reilly چاپ شده و یکی از منابع جدی و پرطرفدار توی حوزه‌ی یادگیری ماشین و یادگیری عمیقه.

این نسخه نسبت به چاپ‌های قبلی کلی تغییر و آپدیت داشته:

- کدها و مثال‌ها همه با آخرین نسخه‌های Scikit-Learn، Keras و TensorFlow 2.x هماهنگ شدن.
- موضوعات جذاب و جدیدی مثل ترنسفورمرها، شبکه‌های مولد تخاصمی (GANs) و یادگیری خودنظارتی اضافه شده.
- توضیحات ساده‌تر و روان‌تر شده تا هم تازه‌کارها بتونن راحت‌تر جلو بیان، هم افراد حرفه‌ای‌تر نکات جدید پیدا کنن.
- تمرین‌ها و کدنویسی‌ها هم با استانداردهای علمی و صنعتی امروز هماهنگ شدن.

این جلدی که الان آماده کردم مربوط به بخش اول کتاب، یعنی مبانی یادگیری ماشین هست. یعنی تمام پایه‌ها و مفاهیم اصلی این حوزه رو پوشش می‌ده. در جلد دوم سراغ شبکه‌های عصبی و یادگیری عمیق می‌ریم تا مجموعه کامل بشه.

امیدوارم این کار بتونه برای علاقه‌مندان و دوستان جامعه علمی و صنعتی کشور مفید باشه و راه رو برای یادگیری و تجربه‌های عملی بیشتر باز کنه 🙏

© 2025 Mohsen Zare. All rights reserved.

پیشگفتار

© 2025 Mohsen Zare. All rights reserved.

سونامی یادگیری ماشین

در سال ۲۰۰۶، جفری هینتون و همکارانش مقاله‌ای منتشر کردند که نشان می‌داد چگونه می‌توان یک شبکه عصبی عمیق را آموزش داد تا با دقتی خیره‌کننده (بیش از ۹۸%) ارقام دست‌نویس را تشخیص دهد. آنها این رویکرد را «یادگیری عمیق (DL: *Deep Learning*)» نامیدند. در آن زمان، آموزش شبکه‌های عصبی عمیق به‌طور گسترده غیرممکن تلقی می‌شد و بسیاری از پژوهشگران از دهه ۱۹۹۰ این ایده را کنار گذاشته بودند. انتشار این مقاله بار دیگر توجه جامعه علمی را به این حوزه جلب کرد و به‌سرعت، موجی از تحقیقات جدید نشان داد که یادگیری عمیق نه‌تنها شدنی است، بلکه می‌تواند به نتایجی برسد که هیچ روش دیگری در «یادگیری ماشین (ML: *Machine Learning*)» حتی امید دستیابی به آن را نداشته باشد—البته با تکیه بر قدرت بالای محاسباتی و حجم عظیم داده‌ها. این شور و اشتیاق به‌زودی به سایر شاخه‌های یادگیری ماشین نیز گسترش یافت.

ده سال بعد، یادگیری ماشین صنعت را متحول کرد و در قلب بسیاری از فناوری‌های پیشرفته امروزی جای گرفت. اکنون این فناوری نتایج جست‌وجوی وب شما را رتبه‌بندی می‌کند، تشخیص گفتار گوشی هوشمندتان را بهبود می‌بخشد، ویدیوها را پیشنهاد می‌دهد و حتی توانسته است قهرمان جهان را در بازی Go شکست دهد. شاید زودتر از آنچه فکر می‌کنید، خودرویتان نیز توسط آن هدایت شود.

© 2025 Mohsen Zare. All rights reserved.

یادگیری ماشین در پروژه‌های شما

طبیعتاً شما هم از دنیای یادگیری ماشین هیجان‌زده هستید و دوست دارید بخشی از این موج باشید!

شاید بخواهید به ربات خانگی‌تان «مغزی» بدهید که بتواند چهره‌ها را تشخیص دهد یا حتی راه رفتن یاد بگیرد.

یا ممکن است شرکتتان حجم زیادی داده در اختیار داشته باشد—از گزارش‌های کاربران و داده‌های مالی گرفته تا اطلاعات تولید، داده‌های حسگرها، آمار تماس‌های تلفنی و گزارش‌های منابع انسانی—و شما حدس بزنید که اگر بدانید چگونه آن‌ها را بررسی کنید، می‌توانید گنجینه‌های پنهانی در میانشان بیابید.

با کمک یادگیری ماشین می‌توان کارهایی مانند موارد زیر و حتی بسیار فراتر از آن انجام داد:

- مشتریان را دسته‌بندی کنید و برای هر گروه، بهترین استراتژی بازاریابی را بیابید.
- بر اساس خریدهای مشتریان مشابه، به هر فرد محصولات مناسب را پیشنهاد دهید.
- تراکنش‌هایی را که احتمال جعلی بودن دارند شناسایی کنید.
- درآمد سال آینده را پیش‌بینی کنید.
- و بسیاری کاربردهای دیگر.

به هر دلیل، تصمیم گرفته‌اید یادگیری ماشین را فرا بگیرید و آن را در پروژه‌های خود به کار بگیرید.

© 2025 Mohsen Zare. All rights reserved.

هدف و رویکرد

این کتاب فرض می‌کند که شما تقریباً هیچ دانشی درباره یادگیری ماشین ندارید. هدف آن ارائه مفاهیم، ابزارها و درک شهودی ای است که برای ساخت برنامه‌هایی با قابلیت «یادگیری از داده‌ها (learning from data)» به آن نیاز دارید.

ما طیف گسترده‌ای از تکنیک‌ها را بررسی خواهیم کرد—از ساده‌ترین و رایج‌ترین روش‌ها، مانند رگرسیون خطی، تا برخی از تکنیک‌های یادگیری عمیق که به طور مداوم در رقابت‌ها برنده می‌شوند. برای این کار، از چارچوب‌های آماده و حرفه‌ای پایتون استفاده خواهیم کرد:

- Scikit-Learn استفاده‌ای بسیار ساده دارد، اما در عین حال بسیاری از الگوریتم‌های یادگیری ماشین را به شکلی کارآمد پیاده‌سازی می‌کند؛ به همین دلیل نقطه شروعی عالی برای ورود به این حوزه است. این کتابخانه در سال ۲۰۰۷ توسط *دیوید کورناپئو* ایجاد شد و اکنون تیمی از پژوهشگران در مؤسسه فرانسوی تحقیقات علوم کامپیوتر و اتوماسیون (Inria) آن را توسعه و هدایت می‌کنند.

- TensorFlow یک کتابخانه پیشرفته‌تر برای محاسبات عددی توزیع‌شده است. این کتابخانه با توزیع پردازش‌ها روی صدها سرور و حتی چندین GPU (واحد پردازش گرافیکی)، امکان آموزش و اجرای بسیار کارآمد شبکه‌های عصبی بزرگ را فراهم می‌کند. تنسورفلو در گوگل توسعه یافت و بسیاری از برنامه‌های یادگیری ماشین در مقیاس بزرگ این شرکت بر پایه آن اجرا می‌شوند. این کتابخانه در نوامبر ۲۰۱۵ به‌صورت متن‌باز منتشر شد و نسخه 2.0 آن در سپتامبر ۲۰۱۹ عرضه گردید.

- Keras یک رابط برنامه‌نویسی سطح‌بالا (API) برای یادگیری عمیق است که فرآیند آموزش و اجرای شبکه‌های عصبی را به‌طور چشمگیری ساده می‌کند. کِراس همراه با تنسورفلو ارائه می‌شود و تمام محاسبات سنگین خود را بر پایه تنسورفلو انجام می‌دهد.

این کتاب رویکردی کاملاً عملی دارد و با استفاده از مثال‌های کاربردی و ملموس، همراه با اندکی توضیح نظری، به پرورش درک شهودی شما از یادگیری ماشین کمک می‌کند.

© 2025 Mohsen Zare. All rights reserved.

مثال‌های کد

تمام نمونه‌کدهای این کتاب متن‌باز هستند و به‌صورت آنلاین در قالب Jupyter Notebookها در آدرس *https://github.com/ageron/handson-ml3* در دسترس قرار دارند. این نوت‌بوک‌ها اسناد تعاملی‌ای هستند که شامل متن، تصویر و قطعه‌کدهای(در این کتاب با زبان پایتون) قابل اجرا می‌شوند. ساده‌ترین و سریع‌ترین راه برای شروع، اجرای این نوت‌بوک‌ها با استفاده از Google Colab است؛ سرویسی رایگان که به شما امکان می‌دهد هر Jupyter Notebook را مستقیماً به‌صورت آنلاین، بدون نیاز به نصب هیچ نرم‌افزاری روی دستگاه، اجرا کنید. تنها چیزی که لازم دارید یک مرورگر وب و یک حساب کاربری گوگل است.

توجه

در این کتاب، فرض بر این است که از Google Colab استفاده می‌کنید، اما من نوت‌بوک‌ها را روی پلتفرم‌های آنلاین دیگری مانند Kaggle و Binder نیز آزمایش کرده‌ام؛ بنابراین در صورت تمایل می‌توانید از آن‌ها هم بهره ببرید. همچنین می‌توانید کتابخانه‌ها و ابزارهای موردنیاز (یا تصویر Docker مخصوص این کتاب) را نصب کرده و نوت‌بوک‌ها را مستقیماً روی دستگاه خود اجرا کنید. دستورالعمل کامل این کار در *https://homl.info/install* موجود است.

© 2025 Mohsen Zare. All rights reserved.

پیش‌نیازها

این کتاب فرض می‌کند که شما تا حدی با برنامه‌نویسی پایتون آشنا هستید. اگر هنوز پایتون را یاد نگرفته‌اید، وب‌سایت *https://learnpython.org* نقطه شروع بسیار مناسبی است. همچنین، آموزش رسمی موجود در *Python.org* نیز منبعی عالی برای یادگیری است.

این کتاب همچنین فرض می‌کند که با کتابخانه‌های علمیِ اصلی پایتون—به‌ویژه NumPy، Pandas و Matplotlib—آشنا هستید. اگر تاکنون با این کتابخانه‌ها کار نکرده‌اید، جای نگرانی نیست؛ یادگیری آن‌ها ساده است و برای هرکدام، یک آموزش اختصاصی تهیه کرده‌ام که می‌توانید به‌صورت آنلاین از طریق *https://homl.info/tutorials* به آن‌ها دسترسی پیدا کنید.

علاوه بر این، اگر می‌خواهید درک کاملی از نحوه کار الگوریتم‌های یادگیری ماشین پیدا کنید (و نه فقط روش استفاده از آن‌ها)، لازم است حداقل آشنایی اولیه‌ای با چند مفهوم ریاضی، به‌ویژه جبر خطی، داشته باشید. به طور مشخص، باید بدانید بردارها و ماتریس‌ها چه هستند و بتوانید برخی عملیات ساده مانند جمع بردارها یا ترانهاده و ضرب ماتریس‌ها را انجام دهید. اگر به یک مرور سریع جبر خطی نیاز دارید (واقعاً پیچیده نیست!)، من آموزشی در *https://homl.info/tutorials* قرار داده‌ام. همچنین، در آن‌جا یک آموزش حساب دیفرانسیل و انتگرال هم خواهید یافت که می‌تواند برای درک نحوه آموزش شبکه‌های عصبی مفید باشد، هرچند برای فهم مفاهیم اصلی کاملاً ضروری نیست. این کتاب گاهی به مفاهیم ریاضی دیگری مانند توابع نمایی و لگاریتم، مقداری نظریه احتمال و برخی مفاهیم پایه آمار اشاره می‌کند، اما هیچ‌یک پیشرفته یا دشوار نیستند. اگر در این موارد به کمک نیاز داشتید، وب‌سایت *https://khanacademy.org* منبعی عالی است که دوره‌های رایگان و باکیفیت زیادی را به صورت آنلاین ارائه می‌دهد.

اگر تاکنون با Jupyter کار نکرده‌اید، فصل ۲ شما را با مراحل نصب و اصول اولیه آن آشنا می‌کند. ژوپیتر ابزاری فوق‌العاده است که حتماً باید در جعبه‌ابزار کاری خود داشته باشید.

اگر با کتابخانه‌های علمی پایتون آشنایی ندارید، Jupyter Notebook ارائه‌شده در این کتاب شامل چند آموزش کاربردی هستند. همچنین، یک آموزش سریع جبر خطی هم در آن‌ها گنجانده شده است.

© 2025 Mohsen Zare. All rights reserved.

نقشه راه

کتاب اصلی در یک جلد منتشر شده است، اما برای سهولت مطالعه، ترجمه‌ی حاضر در دو جلد ارائه می‌شود. جلد اول ترجمه، شامل بخش اول کتاب اصلی، فصل‌های ۱ الی ۹، با عنوان «مبانی یادگیری ماشین»، مباحث زیر را پوشش می‌دهد:

- یادگیری ماشین چیست، چه مسائلی را حل می‌کند، و دسته‌بندی‌ها و مفاهیم اساسی آن
- مراحل معمول یک پروژه یادگیری ماشین
- آموزش مدل با برازش به داده‌ها
- بهینه‌سازی تابع هزینه
- مدیریت، پاکسازی و آماده‌سازی داده‌ها
- انتخاب و مهندسی ویژگی‌ها
- انتخاب مدل و تنظیم هایپرپارامترها با استفاده از اعتبارسنجی متقاطع
- چالش‌های یادگیری ماشین، به‌ویژه کم‌برازش و بیش‌برازش (توازن بایاس/واریانس)
- رایج‌ترین الگوریتم‌های یادگیری شامل: رگرسیون خطی و چندجمله‌ای، رگرسیون لجستیک، k-نزدیک‌ترین همسایه، ماشین‌های بردار پشتیبان، درخت تصمیم، جنگل‌های تصادفی و روش‌های ترکیبی(تجمیعی)
- کاهش ابعاد داده‌های آموزشی برای مقابله با «مُعضَل ابعاد بالا»
- تکنیک‌های یادگیری بدون نظارت، از جمله خوشه‌بندی، تخمین چگالی و تشخیص ناهنجاری

© 2025 Mohsen Zare. All rights reserved.

جلد دوم ترجمه، شامل بخش دوم کتاب اصلی، فصل‌های ۱۰ الی ۱۹، با عنوان «شبکه‌های عصبی و یادگیری عمیق» است و مباحث زیر را پوشش می‌دهد:

- شبکه‌های عصبی چیستند و چه کاربردهایی دارند؟
- ساخت و آموزش شبکه‌های عصبی با استفاده از TensorFlow و Keras
- معرفی مهم‌ترین معماری‌های شبکه عصبی:
  - شبکه‌های عصبی پیش‌خور برای داده‌های جدولی
  - شبکه‌های کانولوشن برای بینایی کامپیوتر
  - شبکه‌های بازگشتی و LSTM برای پردازش توالی
  - رمزگذار-رمزگشاها و ترنسفورمرها برای پردازش زبان طبیعی و کاربردهای دیگر
  - رمزگذارهای خودکار
  - شبکه‌های مولد تخاصمی (GANs) و مدل‌های انتشار برای یادگیری مولد
- تکنیک‌هایی برای آموزش شبکه‌های عصبی عمیق
- ساخت عامل‌هایی با یادگیری تقویتی، که بتوانند از طریق آزمون و خطا استراتژی‌های مؤثر یاد بگیرند (مثلاً ربات در بازی)
- بارگذاری و پیش‌پردازش داده‌های حجیم به شکل مؤثر
- آموزش و استقرار مدل‌های TensorFlow در مقیاس بزرگ

© 2025 Mohsen Zare. All rights reserved.

بخش اول کتاب عمدتاً بر Scikit-Learn تمرکز دارد، در حالی که بخش دوم به کارگیری TensorFlow و Keras را آموزش می‌دهد.

هشدار

خیلی سریع وارد مباحث عمیق نشوید: اگرچه یادگیری عمیق بدون شک یکی از هیجان‌انگیزترین شاخه‌های یادگیری ماشین است، اما بهتر است ابتدا اصول پایه را به خوبی فرا بگیرید. علاوه بر این، بیشتر مسائل را می‌توان با تکنیک‌های ساده‌تر، مانند جنگل‌های تصادفی و روش‌های گروهی (که در بخش اول بررسی شده‌اند)، به‌خوبی حل کرد. یادگیری عمیق بیشتر برای مسائلی مانند تشخیص تصویر، تشخیص گفتار و پردازش زبان طبیعی کاربرد دارد—البته به شرط آنکه داده‌های کافی، قدرت محاسباتی مناسب و صبر کافی در اختیار داشته باشید.

© 2025 Mohsen Zare. All rights reserved.

منابع دیگر

منابع زیادی برای یادگیری ماشین وجود دارد که می‌توانید از آن‌ها استفاده کنید. برای مثال، دوره «یادگیری ماشین» اندرو اِنگ در سایت Coursera فوق‌العاده است، هرچند باید زمان قابل‌توجهی برای آن کنار بگذارید.

وب‌سایت‌های جذاب زیادی هم در این حوزه هستند، مثل راهنمای کاربر بسیار خوب Scikit-Learn. همچنین ممکن است از آموزش‌های تعاملی عالی سایت Dataquest یا وبلاگ‌های مرتبط با یادگیری ماشین، مانند وبلاگ‌های موجود در Quora، لذت ببرید. در نهایت، وب‌سایت Deep Learning هم مجموعه‌ای ارزشمند از منابع آموزشی برای یادگیری بیشتر فراهم کرده است.

البته کتاب‌های مقدماتی زیادی هم در زمینه یادگیری ماشین منتشر شده‌اند، از جمله:

- *جوئل گروس* – «علم داده از ابتدا» (انتشارات O'Reilly): این کتاب مبانی یادگیری ماشین را توضیح می‌دهد و تعدادی از الگوریتم‌های مهم را با استفاده از پایتون خالص، همان‌طور که از نامش پیداست، «از صفر» پیاده‌سازی می‌کند.
- *استیون مارسلند* – «یادگیری ماشین: یک دیدگاه الگوریتمی» (ویرایش دوم، Chapman & Hall): این کتاب مقدمه‌ای جامع بر یادگیری ماشین ارائه می‌دهد و دامنه وسیعی از موضوعات را با جزئیات بررسی می‌کند. مثال‌های کد آن به زبان پایتون نوشته شده‌اند و الگوریتم‌ها را از پایه، اما با بهره‌گیری از کتابخانه NumPy، پیاده‌سازی می‌کند.
- *سباستین راشکا* – «یادگیری ماشین با پایتون» (ویرایش سوم، انتشارات Packt): این کتاب مقدمه‌ای بسیار خوب بر یادگیری ماشین ارائه می‌دهد و از کتابخانه‌های متن‌باز پایتون مانند Pylearn2 و Theano برای پیاده‌سازی الگوریتم‌ها استفاده می‌کند.

© 2025 Mohsen Zare. All rights reserved.

- *فرانسوا شوله* – «یادگیری عمیق با پایتون» (ویرایش دوم، انتشارات Manning): کتابی بسیار عملی و روان که طیف گسترده‌ای از مباحث را شفاف و مختصر توضیح می‌دهد. همان‌طور که از خالق کتابخانه‌ی محبوب Keras انتظار می‌رود، تمرکز اصلی کتاب بر مثال‌های کدنویسی است تا فرمول‌ها و اثبات‌های ریاضی.

- *آندری بورکوف* – «کتاب صد صفحه‌ای یادگیری ماشین» (خودانتشار): اثری کوتاه اما پربار که با زبانی ساده و روان، دامنه گسترده‌ای از مفاهیم یادگیری ماشین را توضیح می‌دهد و در عین حال، از پیچیدگی‌های غیرضروری و فرمول‌های ریاضی سنگین دوری می‌کند.

- *یاسر س. ابومصطفی، مالک مگدون-اسماعیل و هسوان-تین لین* – «یادگیری از داده‌ها» (AMLBook): این کتاب رویکردی نسبتاً نظری به یادگیری ماشین دارد و بینش‌های عمیقی، به‌ویژه درباره توازن بایاس و واریانس (مراجعه کنید به فصل ۴)، ارائه می‌دهد.

- *استوارت راسل و پیتر نورویگ* – «هوش مصنوعی: رویکردی مدرن» (ویرایش چهارم، انتشارات Pearson): این کتاب جامع و ارزشمند، مباحث گسترده‌ای از جمله یادگیری ماشین را پوشش می‌دهد و به درک عمیق‌تر مفاهیم هوش مصنوعی و یادگیری ماشین کمک زیادی می‌کند.

- *جرمی هاوارد و سیلوین گاگر* – «یادگیری عمیق برای برنامه‌نویسان با fastai و PyTorch» (انتشارات O'Reilly): این کتاب مقدمه‌ای بسیار واضح و کاربردی درباره یادگیری عمیق است که آموزش‌های خود را با استفاده از کتابخانه‌های fastai و PyTorch ارائه می‌دهد.

در نهایت، یکی از بهترین روش‌ها برای یادگیری، شرکت در وب‌سایت‌های مسابقات یادگیری ماشین مانند Kaggle.com است. این پلتفرم به شما فرصت می‌دهد مهارت‌هایتان را روی مسائل واقعی تمرین کنید و از کمک و تجربیات برخی از بهترین متخصصان این حوزه بهره‌مند شوید.

© 2025 Mohsen Zare. All rights reserved.

بخش اول. مبانی یادگیری ماشین

© 2025 Mohsen Zare. All rights reserved.

فصل ۱. چشم‌انداز یادگیری ماشین

چند سال پیش، اگر گوشی را برمی‌داشتید و از آن می‌پرسیدید راه خانه کجاست، شاید مردم شما را نادیده می‌گرفتند و حتی به سلامت عقل‌تان شک می‌کردند. اما حالا یادگیری ماشین دیگر داستانی علمی‌تخیلی نیست؛ میلیاردها نفر هر روز از آن بهره می‌برند. در واقع، این فناوری دهه‌هاست که در برخی برنامه‌های خاص مانند «تشخیصِ نوری کاراکتر (OCR: Optical Character Recognition)» استفاده می‌شود. اولین برنامه یادگیریِ ماشینی که واقعاً به جریان اصلی تبدیل شد و زندگی صدها میلیون نفر را بهبود بخشید، در دهه ۱۹۹۰ به‌طور گسترده استفاده شد: «فیلتر اسپم». این برنامه شاید یک ربات خودآگاه نباشد، اما از نظر فنی یادگیری ماشین محسوب می‌شود؛ در واقع آن‌قدر خوب یاد گرفته که حالا دیگر به ندرت لازم است شما ایمیلی را به عنوان هرزنامه علامت‌گذاری کنید. پس از آن، صدها برنامه یادگیری ماشین به‌وجود آمد که اکنون به‌صورت پنهان و مؤثر، پشت صدها محصول و قابلیت مختلف قرار دارند و هر روز بارها و بارها از آن‌ها استفاده می‌کنید؛ مانند پیام‌های صوتی، ترجمه خودکار، جستجوی تصویر، پیشنهادهای خرید و بسیاری کاربردهای دیگر.

یادگیری ماشین از کجا شروع می‌شود و تا کجا ادامه پیدا می‌کند؟ معنای واقعی «یادگیری» برای یک ماشین چیست؟ مثلاً اگر تمام مقالات ویکی‌پدیا را دانلود کنم، آیا کامپیوتر من واقعاً چیزی یاد گرفته؟ آیا ناگهان باهوش‌تر شده؟ در این فصل، ابتدا به این پرسش‌ها پاسخ می‌دهیم و توضیح می‌دهیم یادگیری ماشین چیست و چرا ممکن است بخواهید از آن استفاده کنید.

© 2025 Mohsen Zare. All rights reserved.

قبل از اینکه به دنیای گسترده یادگیری ماشین وارد شویم، ابتدا نگاهی کلی به نقشه این حوزه می‌اندازیم و با بخش‌های اصلی و مهم‌ترین مفاهیم آن آشنا می‌شویم:

- یادگیری تحت نظارت در مقابل یادگیری بدون نظارت و انواع مختلف آن،
- یادگیری آنلاین در برابر یادگیری دسته‌ای،
- یادگیری مبتنی بر نمونه در مقابل یادگیری مبتنی بر مدل.

پس از آن، گردش کار یک پروژه معمولی یادگیری ماشین–مجموعه مراحل از جمع‌آوری داده، پیش‌پردازش، انتخاب مدل، آموزش، ارزیابی تا استقرار مدل–را بررسی می‌کنیم، چالش‌های اصلی پیش روی شما را مطرح می‌کنیم و در نهایت روش‌های ارزیابی و بهینه‌سازی یک سیستم یادگیری ماشین را یاد می‌گیریم.

این فصل مفاهیم و اصطلاحات پایه‌ای زیادی را معرفی می‌کند که هر دانشمند داده‌ای باید به خوبی بشناسد. این یک مرور کلی و سطح بالا است (تنها فصلی که کدهای زیادی ندارد) و همه چیز نسبتاً ساده باقی می‌ماند. هدف من این است که قبل از پیش‌روی در کتاب، مطمئن شوم همه مفاهیم برای شما کاملاً روشن شده‌اند.

پس یک فنجان قهوه بردارید و آماده شوید تا شروع کنیم!

© 2025 Mohsen Zare. All rights reserved.

یادگیری ماشین چیست؟

یادگیری ماشین، علم و هنر برنامه‌نویسی کامپیوترها برای «یادگیری از داده‌ها» است.

یک تعریف کمی کلی‌تر این است:

«یادگیری ماشین حوزه‌ای است که به کامپیوترها این امکان را می‌دهد بدون برنامه‌نویسی صریح، یاد بگیرند.»

—آرتور ساموئل، ۱۹۵۹

یک تعریف مهندسی‌محورتر این است:

«یک برنامه کامپیوتری وقتی می‌گوییم یاد گرفته است که با استفاده از تجربهٔ E در انجام یک مسئلهٔ T ، عملکرد آن برنامه در آن مسئله، که با معیار P سنجیده می‌شود، بهبود یابد.»

—تام میچل، ۱۹۹۷

فیلتر اسپم شما یک برنامه یادگیری ماشین است که با دریافت نمونه‌هایی از ایمیل‌های اسپم (که کاربران قبلاً علامت زده‌اند) و ایمیل‌های معمولی (که به آن‌ها «ham» نیز می‌گویند)، یاد می‌گیرد چگونه ایمیل‌های اسپم را شناسایی و علامت‌گذاری کند. نمونه‌هایی که سیستم برای یادگیری استفاده می‌کند، «مجموعه آموزشی($training\ set$)» نامیده می‌شوند و هر نمونه آموزشی در این مجموعه «$instance$» یا «$sample$» خوانده می‌شود. بخشی از سیستم یادگیری ماشین که وظیفه یادگیری و پیش‌بینی را بر عهده دارد، «مدل ($model$)» نامیده می‌شود. شبکه‌های عصبی و جنگل‌های تصادفی، نمونه‌هایی از مدل‌های یادگیری ماشین هستند.

در این مثال، مسئله (T) همان شناسایی و علامت‌گذاری هرزنامه‌ها در ایمیل‌های جدید است. تجربه (E) همان داده‌های آموزشی است که سیستم از آن یاد می‌گیرد، و معیار عملکرد (P) باید به‌طور دقیق تعریف شود. برای نمونه، می‌توان از نسبت ایمیل‌هایی که به‌درستی طبقه‌بندی شده‌اند به کل ایمیل‌ها

© 2025 Mohsen Zare. All rights reserved.

استفاده کرد. این معیار عملکرد را «دقت (Accuracy)» می‌نامند، که یکی از شاخص‌های رایج در مسائل طبقه‌بندی است.

اگر فقط یک نسخه‌ی کامل از تمام مقالات ویکی‌پدیا را روی کامپیوتر خود دانلود کنید، دستگاه شما قطعاً حجم عظیمی از داده‌ها خواهد داشت. اما این به‌تنهایی باعث نمی‌شود عملکرد بهتری در هیچ کاری داشته باشد؛ چون این کار هنوز به معنای یادگیری ماشین نیست.

© 2025 Mohsen Zare. All rights reserved.

چرا از یادگیری ماشین استفاده کنیم؟

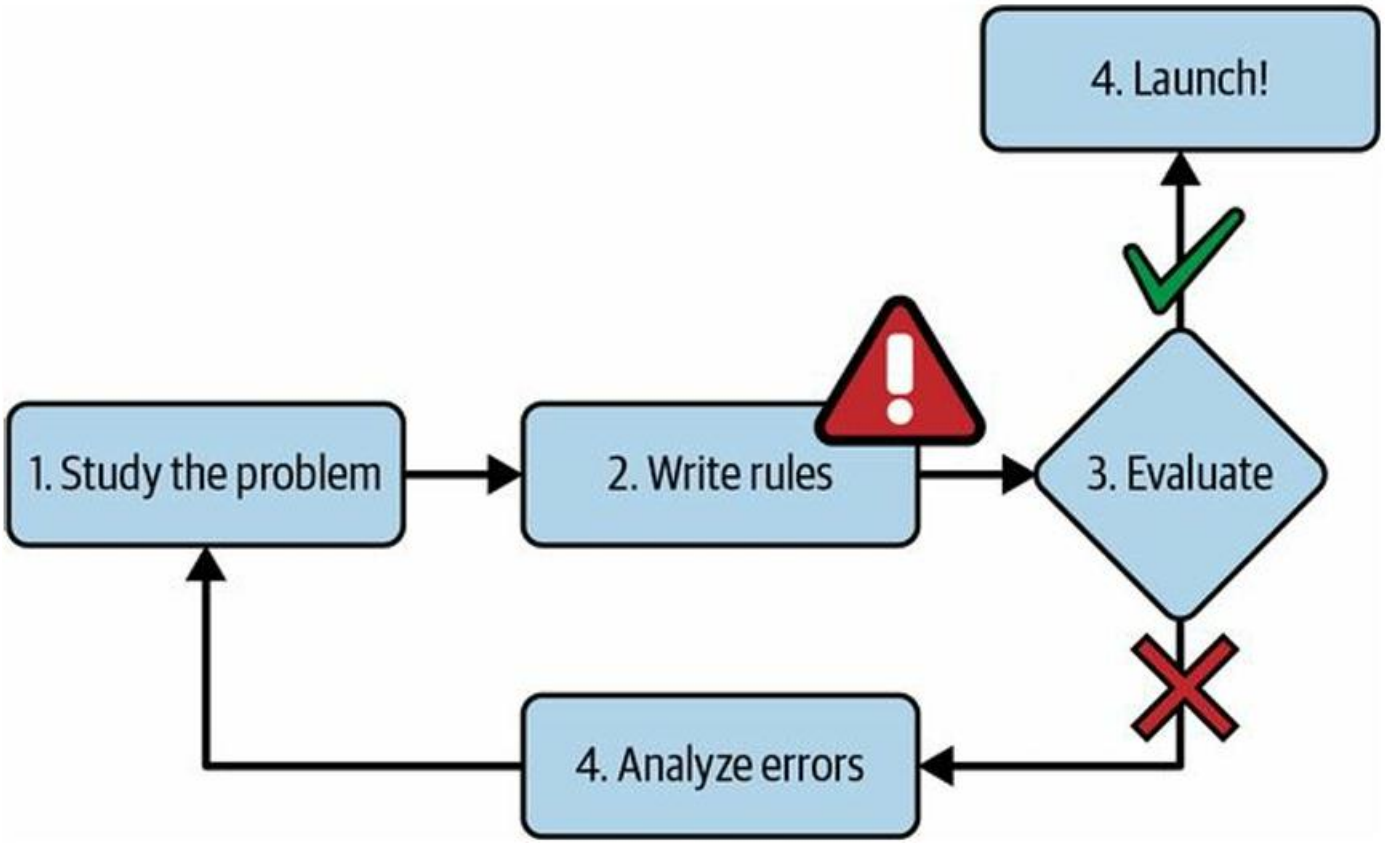
بیایید ببینیم چگونه می‌توان با روش‌های سنتی برنامه‌نویسی یک فیلتر اسپم ساخت (شکل ۱-۱):

۱. ابتدا ویژگی‌های معمول هرزنامه‌ها را شناسایی می‌کنید. مثلاً متوجه می‌شوید که کلماتی مثل «4U»، «کارت اعتباری: credit card»، «رایگان: free» و «شگفت‌انگیز: amazing» در عنوان-های ایمیل‌های اسپم زیاد دیده می‌شوند. همچنین ممکن است الگوهایی در نام فرستنده، متن ایمیل یا بخش‌های دیگر پیدا کنید.

۲. سپس برای هر الگوی شناسایی شده، یک الگوریتم خاص می‌نویسید که وقتی این الگوها را پیدا کند، ایمیل را به عنوان هرزنامه علامت‌گذاری کند.

۳. در نهایت برنامه را آزمایش می‌کنید و مراحل اول و دوم را تا زمانی که عملکرد آن قابل قبول شود، تکرار می‌کنید.

© 2025 Mohsen Zare. All rights reserved.

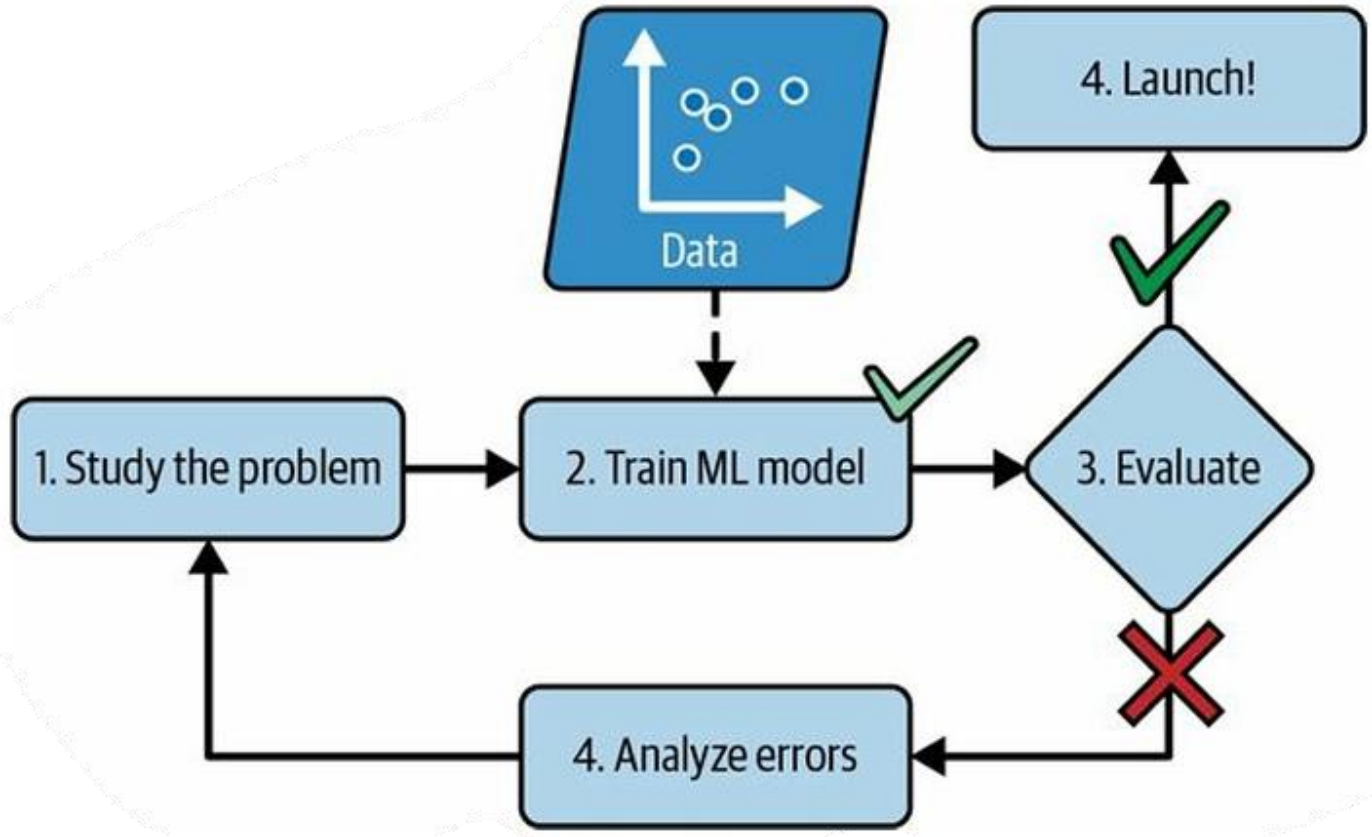


شکل ۱-۱. روش سنتی

از آنجا که این مسئله پیچیده است، برنامه شما به احتمال زیاد به مجموعه‌ای طولانی و پیچیده از قوانین تبدیل می‌شود که نگهداری و به‌روزرسانی آن بسیار دشوار خواهد بود.

در مقابل، فیلتر اسپمی که بر پایه تکنیک‌های یادگیری ماشین ساخته شده باشد، به‌صورت خودکار می‌آموزد که کدام کلمات و عبارات بهترین شاخص‌ها برای تشخیص اسپم هستند. این یادگیری از طریق شناسایی الگوهای غیرمعمول و پرتکرار در نمونه‌های اسپم نسبت به نمونه‌های عادی انجام می‌شود (شکل ۱-۲). چنین برنامه‌ای معمولاً بسیار کوتاه‌تر، نگهداری آن آسان‌تر و اغلب دقیق‌تر است.

© 2025 Mohsen Zare. All rights reserved.

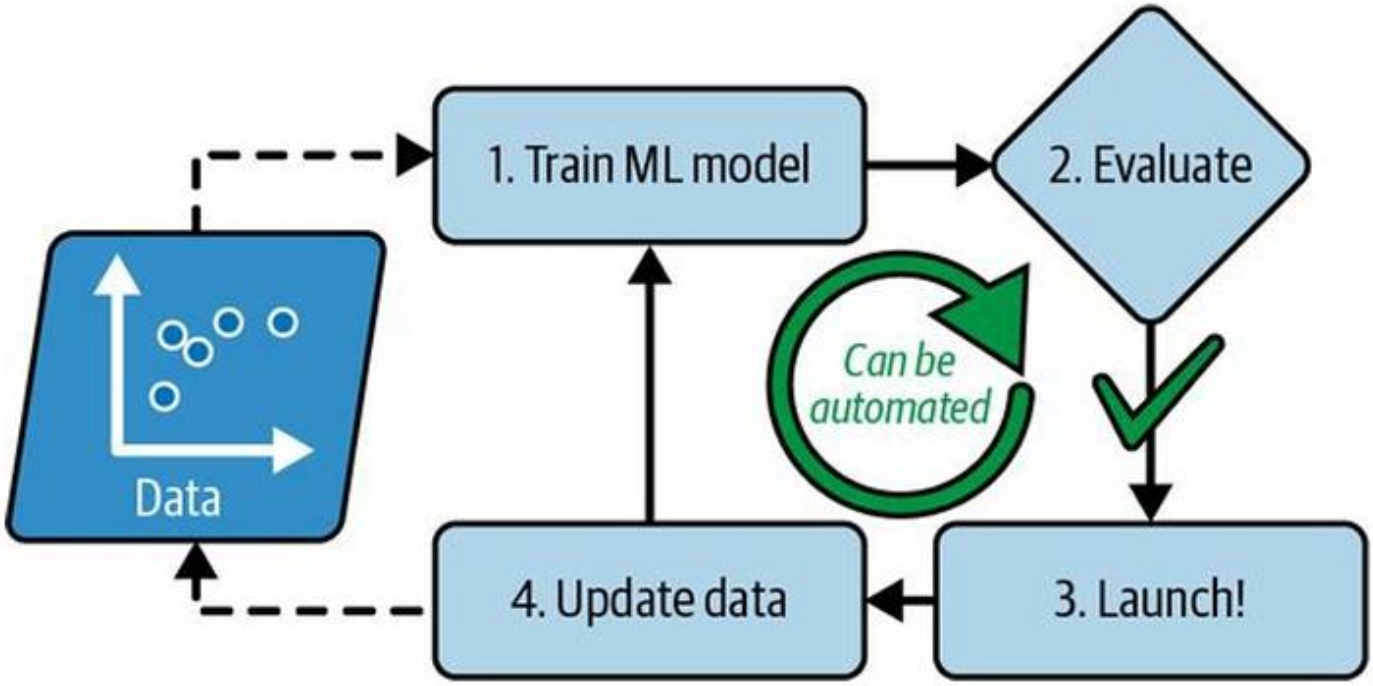


شکل ۲-۱. روش یادگیری ماشین

اگر اِسپَمرها متوجه شوند که همه ایمیل‌هایی که شامل عبارت «4U» هستند، مسدود می‌شوند، ممکن است به جای آن شروع به استفاده از عبارت «For U» کنند. در این حالت، فیلتر اسپمی که بر اساس روش‌های سنتی برنامه‌نویسی ساخته شده، باید به‌روزرسانی شود تا بتواند ایمیل‌های حاوی «For U» را نیز شناسایی و علامت‌گذاری کند. اگر اسپمرها همچنان به دور زدن فیلتر ادامه دهند، شما مجبور خواهید بود همیشه قوانین جدیدی بنویسید و برنامه را به‌روز نگه دارید.

در مقابل، فیلتر اسپمی که مبتنی بر تکنیک‌های یادگیری ماشین است، به‌طور خودکار متوجه می‌شود که عبارت «For U» به طور غیرمعمولی در اسپم‌هایی که کاربران علامت‌گذاری کرده‌اند، تکرار شده است و بدون نیاز به دخالت شما، شروع به علامت‌گذاری این ایمیل‌ها می‌کند (شکل ۱-۳).

© 2025 Mohsen Zare. All rights reserved.



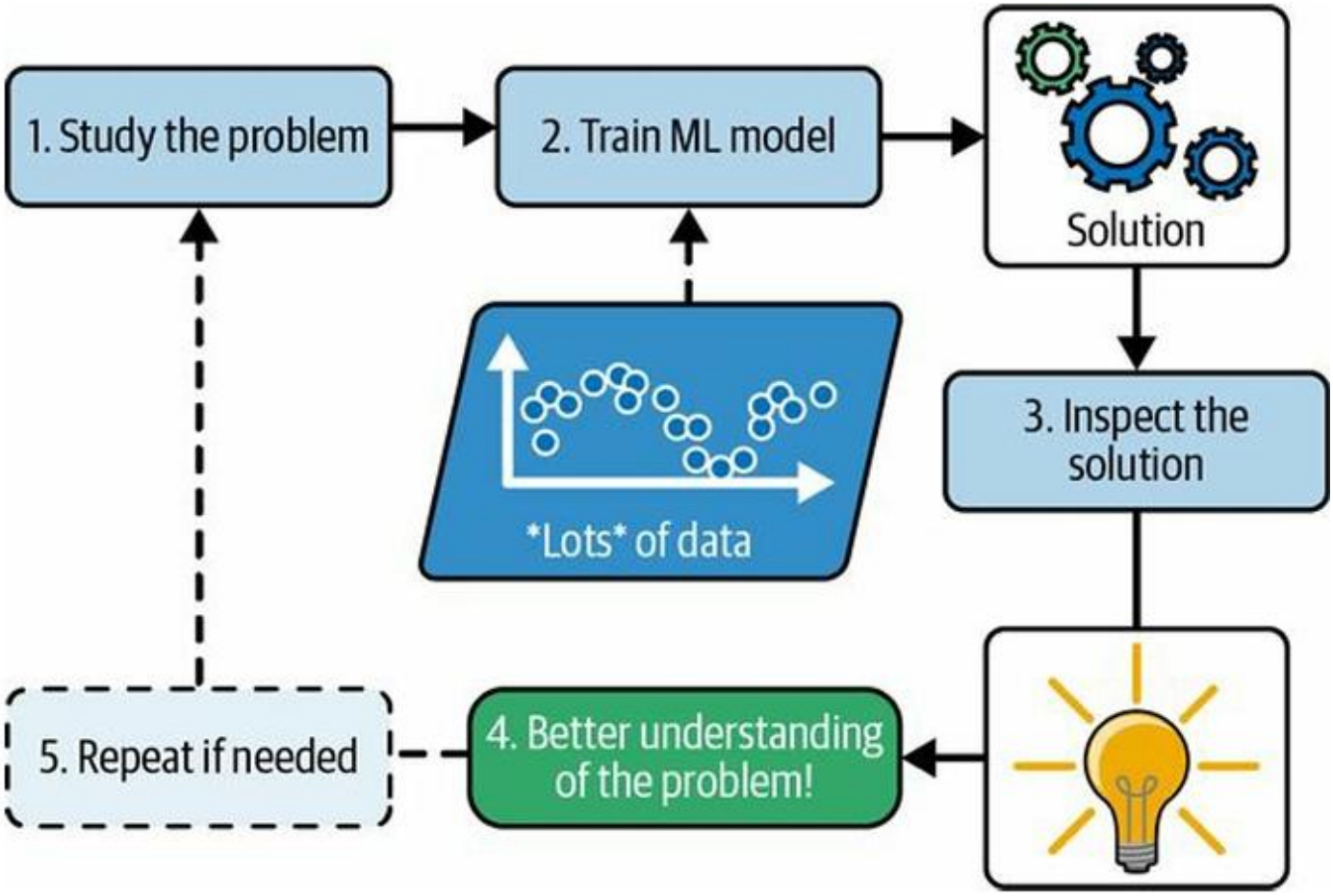
شکل ۳-۱. سازگاری خودکار با تغییرات

یکی دیگر از زمینه‌هایی که یادگیری ماشین در آن برجسته است، مسائلی است که یا برای روش‌های سنتی بسیار پیچیده‌اند یا الگوریتم مشخصی برای حل آن‌ها وجود ندارد. به عنوان مثال، تشخیص گفتار را در نظر بگیرید. فرض کنید می‌خواهید برنامه‌ای بسازید که بتواند کلمات «one» و «two» را تشخیص دهد. ممکن است متوجه شوید که کلمه «two» با صدای زیر «t» شروع می‌شود، پس می‌توانید الگوریتمی بنویسید که شدت این صدای زیر را اندازه‌گیری کند و از آن برای تشخیص این دو کلمه استفاده کنید. اما این روش به وضوح برای هزاران کلمه‌ای که توسط میلیون‌ها فرد با لهجه‌ها و زبان‌های مختلف در محیط‌های پر سر و صدا تلفظ می‌شوند، قابل اجرا نیست. بهترین راه‌حل (حداقل در حال حاضر) این است که الگوریتمی بسازید که با استفاده از نمونه‌های فراوانِ ضبط شده برای هر کلمه، به صورت خودکار یاد بگیرد.

در نهایت، یادگیری ماشین می‌تواند به خود انسان‌ها در فرآیند یادگیری کمک کند (شکل ۱-۴). مدل‌های یادگیری ماشین را می‌توان مورد بررسی قرار داد تا ببینیم چه چیزهایی آموخته‌اند—هرچند در برخی مدل‌ها این کار ممکن است چالش‌برانگیز باشد. برای مثال، پس از آموزشِ کافیِ یک فیلتر اسپم، می‌توان به سادگی آن را تحلیل کرد تا فهرستی از کلمات و ترکیباتی که بهترین پیش‌بینی‌کننده‌های اسپم هستند را به دست آورد. گاهی اوقات این روند باعث کشف همبستگی‌های غیرمنتظره یا روندهای

© 2025 Mohsen Zare. All rights reserved.

تازه‌ای می‌شود که به فهم بهتر مسئله کمک می‌کند. کاوش در حجم بالای داده‌ها برای کشف الگوهای نهفته، «داده‌کاوی ($data\ mining$)» نام دارد و یادگیری ماشین در این زمینه عملکردی فوق‌العاده دارد.



شکل ۱-۴. یادگیری ماشین می‌تواند به انسان‌ها در یادگیری کمک کند

به طور خلاصه، یادگیری ماشین در موارد زیر عملکرد بسیار خوبی دارد:

- مسائلی که راه‌حل‌های سنتی برای آن‌ها نیازمند تنظیمات دقیق فراوان یا فهرست‌های طولانی از قوانین است—در چنین مواردی، یک مدل یادگیری ماشین معمولاً می‌تواند کد را ساده‌تر کرده و عملکرد بهتری ارائه دهد.
- مسائلی که پیچیدگی زیادی دارند و روش‌های سنتی راه‌حل‌های مناسبی برای آن‌ها ندارند—در این موارد، بهترین الگوریتم‌های یادگیری ماشین ممکن است بتوانند راه‌حلی پیدا کنند.

© 2025 Mohsen Zare. All rights reserved.

- محیط‌های پویا و در حال تغییر—زیرا سیستم‌های یادگیری ماشین به سادگی با داده‌های جدید دوباره آموزش داده شده و همواره به‌روز نگه داشته می‌شوند.
- استخراج بینش(اطلاعات مفید و قابل فهم از داده‌ها) و شناخت بهتر مسائل پیچیده از دل حجم انبوهی از داده‌ها.

© 2025 Mohsen Zare. All rights reserved.

نمونه‌هایی از کاربردهای یادگیری ماشین

بیایید چند نمونه ملموس از مسائل(task های) یادگیری ماشین را بررسی کنیم و تکنیک‌هایی را که برای انجام آن‌ها به کار می‌روند، معرفی کنیم:

تجزیه و تحلیل تصاویر محصولات در خط تولید برای طبقه‌بندی خودکار

این مسئله نمونه‌ای از طبقه‌بندی تصویر است که معمولاً با استفاده از شبکه‌های عصبی کانولوشن (CNNها؛ فصل ۱۴ را ببینید) و گاهی با کمک ترنسفورمرها (فصل ۱۶) انجام می‌شود.

تشخیص تومورها در اسکن‌های مغزی

این مسئله مربوط به بخش‌بندی معنایی تصویر است، جایی که هر پیکسل در تصویر طبقه‌بندی می‌شود تا موقعیت و شکل دقیق تومورها مشخص شود. این کار معمولاً با استفاده از CNNها یا ترنسفورمرها انجام می‌گیرد.

طبقه‌بندی خودکار مقالات خبری

این بخش از پردازش زبان طبیعی (NLP) به ویژه طبقه‌بندی متن است. معمولاً از شبکه‌های عصبی بازگشتی (RNN) و CNNها برای این منظور استفاده می‌شود، اما ترنسفورمرها عملکرد بهتری دارند (فصل ۱۶).

علامت‌گذاری خودکار نظرات توهین‌آمیز در انجمن‌های گفتگو

این نیز نوعی طبقه‌بندی متن است که از ابزارهای پردازش زبان طبیعی (NLP) بهره می‌برد.

خلاصه‌سازی خودکار اسناد طولانی

شاخه‌ای از پردازش زبان طبیعی است که به خلاصه‌سازی متن اختصاص دارد و مجدداً از همان ابزارهای NLP استفاده می‌کند.

© 2025 Mohsen Zare. All rights reserved.

ایجاد یک چت‌بات یا دستیار شخصی

این پروژه شامل اجزای متعددی از NLP است، از جمله درک زبان طبیعی (NLU) و ماژول‌های پاسخ به پرسش.

پیش‌بینی درآمد شرکت شما در سال آینده، بر اساس بسیاری از معیارهای عملکرد

این یک مسئله رگرسیون (پیش‌بینی مقادیر پیوسته) است که می‌توان آن را با مدل‌های مختلفی مانند رگرسیون خطی یا رگرسیون چندجمله‌ای (فصل ۴)، ماشین بردار پشتیبان برای رگرسیون (فصل ۵)، جنگل تصادفی رگرسیونی (فصل ۷) یا شبکه‌های عصبی مصنوعی (فصل ۱۰) حل کرد. اگر بخواهید توالی زمانی شاخص‌های گذشته را نیز در نظر بگیرید، می‌توانید از شبکه‌های بازگشتی (RNN)، CNNها یا ترنسفورمرها استفاده کنید (فصل‌های ۱۵ و ۱۶).

واکنش برنامه به دستورات صوتی

این مسئله در واقع همان تشخیص گفتار است که نیازمند پردازش نمونه‌های صوتی می‌باشد. از آنجا که این نمونه‌ها توالی‌های طولانی و پیچیده‌ای هستند، معمولاً با شبکه‌های بازگشتی (RNNها)، شبکه‌های عصبی کانولوشن (CNNها) یا ترنسفورمرها پردازش می‌شوند (فصل‌های ۱۵ و ۱۶).

تشخیص کلاهبرداری در کارت‌های اعتباری

این یک مسئله‌ی تشخیص ناهنجاری است که می‌توان با استفاده از جنگل‌های ایزوله، مدل‌های مخلوط گاوسی (فصل ۹) یا رمزگذارهای خودکار (فصل ۱۷) آن را حل کرد.

تقسیم‌بندی مشتریان بر اساس الگوهای خرید برای طراحی استراتژی‌های بازاریابی متفاوت

این مسئله در حوزه خوشه‌بندی قرار دارد و معمولاً با روش‌هایی مانند k-means، DBSCAN و سایر الگوریتم‌ها انجام می‌شود (فصل ۹).

نمایش داده‌های پیچیده و با ابعاد زیاد در نموداری ساده و قابل فهم

این مسئله مربوط به تجسم داده‌ها است که اغلب با تکنیک‌های کاهش ابعاد ترکیب می‌شود (فصل ۸).

© 2025 Mohsen Zare. All rights reserved.

پیشنهاد محصول به مشتری بر اساس خریدهای قبلی

این یک سیستم توصیه‌گر است. در این روش، اطلاعات خرید گذشته و دیگر داده‌های مرتبط با مشتری به یک شبکه عصبی مصنوعی (فصل ۱۰) داده می‌شود تا محتمل‌ترین محصول بعدی را پیش‌بینی کند. این شبکه عصبی معمولاً با تحلیل توالی خریدهای گذشته همه مشتریان آموزش داده می‌شود.

ساخت ربات هوشمند برای بازی‌ها

این کار اغلب با یادگیری تقویتی (RL : Reinforcement Learning ؛ فصل ۱۸) انجام می‌شود. در این شاخه از یادگیری ماشین، عامل‌ها (مثل ربات‌ها) یاد می‌گیرند اقداماتی را انتخاب کنند که پاداش طولانی‌مدت آنها را ماکسیمُم کند. به عنوان مثال، یک ربات در بازی ممکن است هر بار که بازیکن امتیاز از دست می‌دهد، پاداش دریافت کند. برنامه مشهور AlphaGo که قهرمان جهان بازی Go را شکست داد، با استفاده از RL ساخته شده است.

این فهرست قابل گسترش است، اما هدف این است که به شما نشان دهیم یادگیری ماشین تا چه حد گسترده و پیچیده است و چه طیف متنوعی از وظایف و تکنیک‌ها را شامل می‌شود.

© 2025 Mohsen Zare. All rights reserved.

انواع سیستم‌های یادگیری ماشین

سیستم‌های یادگیری ماشین بسیار متنوع‌اند و برای درک بهتر، معمولاً آنها را بر اساس چند معیار کلی دسته‌بندی می‌کنند:

- نحوه نظارت در فرایند آموزش:
  - سیستم‌های نظارت‌شده (Supervised)
  - بدون نظارت (Unsupervised)
  - نیمه‌نظارت‌شده (Semi-supervised)
  - خودنظارتی (Self-supervised)
  - و سایر روش‌ها
- توانایی یادگیری تدریجی در حین آموزش: یعنی آیا سیستم می‌تواند به صورت یادگیری آنلاین (Online Learning) عمل کند یا فقط به صورت یادگیری دسته‌ای (Batch Learning) کار می‌کند.
- رویکرد یادگیری: آیا سیستم صرفاً نقاط داده جدید را با نمونه‌های شناخته شده مقایسه می‌کند (یادگیری مبتنی بر نمونه یا Instance-based Learning) یا اینکه با شناسایی الگوها در داده‌های آموزشی، یک مدل پیش‌بینی می‌سازد (یادگیری مبتنی بر مدل یا Model-based Learning)؛ درست مانند کاری که دانشمندان انجام می‌دهند.

این دسته‌بندی‌ها محدود به هم نیستند و می‌توان آنها را به دلخواه و به صورت ترکیبی استفاده کرد. برای مثال، یک فیلتر اسپم پیشرفته که از شبکه عصبی عمیق استفاده می‌کند و با نمونه‌های اسپم و غیر اسپم که توسط انسان برچسب‌گذاری شده‌اند آموزش دیده و همزمان به صورت آنلاین یاد می‌گیرد، ترکیبی از یک سیستم نظارت‌شده، آنلاین و مبتنی بر مدل است. بیایید کمی دقیق‌تر به هر یک از این معیارها بپردازیم.

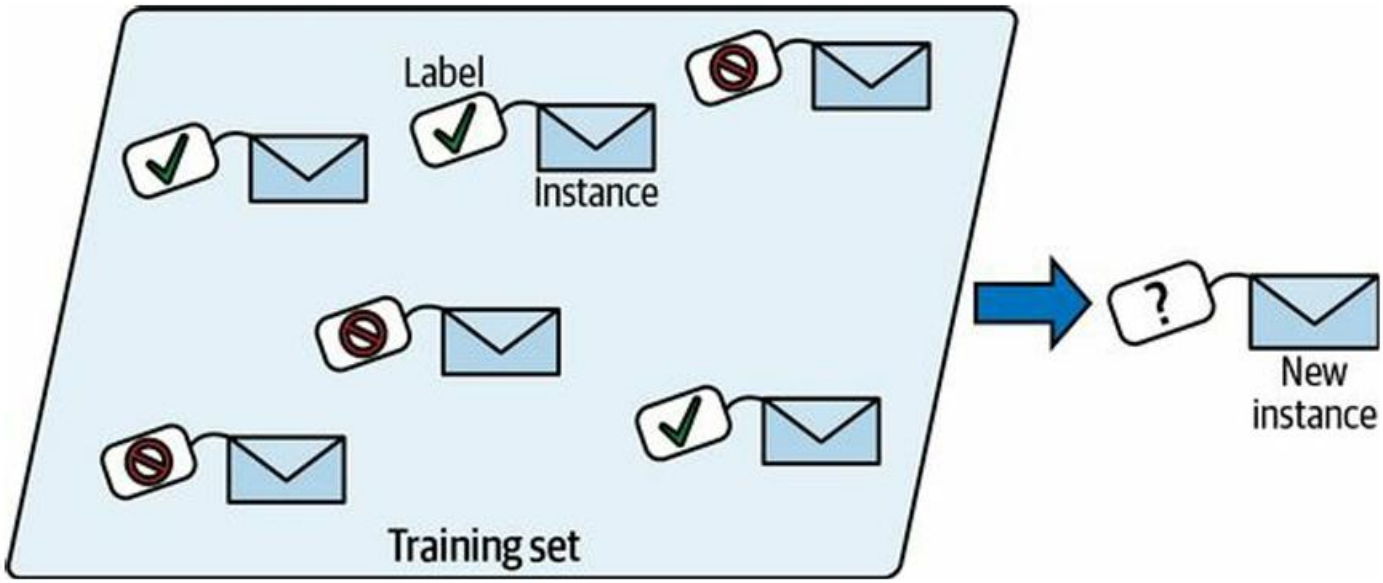
© 2025 Mohsen Zare. All rights reserved.

نظارت بر آموزش

سیستم‌های یادگیری ماشین (ML systems) را می‌توان بر اساس نوع و میزان نظارتی که در طول فرایند آموزش دریافت می‌کنند، دسته‌بندی کرد. گرچه دسته‌های متعددی وجود دارد، اما ما در اینجا به مهم‌ترین آنها می‌پردازیم: یادگیری نظارت‌شده، یادگیری بدون نظارت، یادگیری خودنظارتی، یادگیری نیمه‌نظارت‌شده و یادگیری تقویتی.

یادگیری نظارت‌شده

در «یادگیری نظارت‌شده (*supervised learning*)»، مجموعه‌داده‌ای (*dataset*) که به الگوریتم آموزش داده می‌شود، شامل نمونه‌هایی است که هر کدام «برچسب(*label*)» مشخص و درست خود را دارند (شکل ۱-۵). این برچسب‌ها نقش پاسخ یا راه‌حل درست را ایفا می‌کنند.



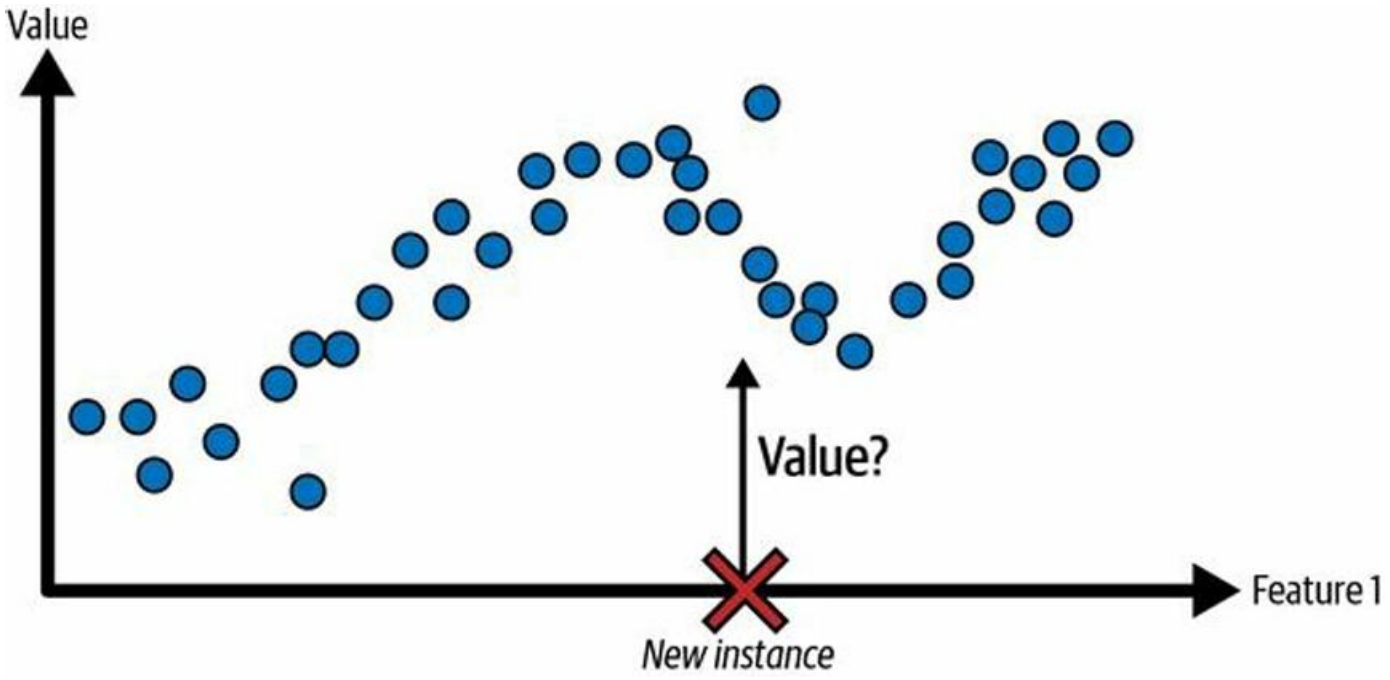
شکل ۱-۵. نمونه‌ای از مجموعه داده‌های آموزشی برچسب‌گذاری شده برای تشخیص هرزنامه(نمونه‌ای بارز از یادگیری نظارت‌شده)

© 2025 Mohsen Zare. All rights reserved.

یکی از کاربردهای معمول یادگیری نظارت‌شده، «طبقه‌بندی یا دسته‌بندی (*classification*)» است. برای مثال، فیلتر اسپم دقیقا همین کار را انجام می‌دهد: سیستم با استفاده از تعداد زیادی ایمیل نمونه به همراه برچسب کلاس آن‌ها (اسپم یا عادی) آموزش می‌بیند و سپس یاد می‌گیرد چگونه ایمیل‌های جدید را در این دسته‌ها قرار دهد.

یکی دیگر از مسائل (یا تَسک‌های) رایج یادگیری نظارت‌شده، پیش‌بینی یک مقدار عددی «هدف (*target*)» مانند قیمت یک خودرو است، بر اساس مجموعه‌ای از «ویژگی‌ها (*features*)» مانند مسافت پیموده شده، سن، برند و غیره. این نوع مسئله «رگرسیون (*regression*)» نامیده می‌شود (شکل ۱-۶). برای آموزش مدل، باید نمونه‌های متعددی از خودروها را به همراه ویژگی‌ها و مقادیر هدفشان (یعنی قیمت‌ها) در اختیار سیستم قرار دهید.

توجه داشته باشید که برخی مدل‌های رگرسیون قابلیت استفاده در طبقه‌بندی را نیز دارند و بالعکس. برای مثال، «رگرسیونِ لُجستیک (*logistic regression*)» معمولاً برای طبقه‌بندی به کار می‌رود، چرا که خروجی آن می‌تواند نشان‌دهنده‌ی احتمال تعلق یک نمونه به یک کلاس مشخص باشد؛ مثلاً احتمال ۲۰% برای اینکه یک ایمیل اسپم باشد.



شکل ۱-۶. نمونه‌ای از مسئله رگرسیون، که در آن هدف، پیش‌بینی یک مقدار عددی بر اساس یک یا چند ویژگی ورودی است (در برخی موارد، چندین مقدار خروجی نیز ممکن است وجود داشته باشد).

© 2025 Mohsen Zare. All rights reserved.

نکته

کلمات «هدف(target)» و «برچسب(label)» معمولاً در مسائل یادگیری نظارت‌شده به‌جای یکدیگر به‌کار می‌روند، اما معمولاً «هدف» در مسائل رگرسیون و «برچسب» در مسائل طبقه‌بندی بیشتر استفاده می‌شود. همچنین، «ویژگی‌ها (features)» گاهی با واژه‌های «پیش‌بینی‌کننده‌ها (*predictors*)» یا «ویژگی‌ها (*attributes*)»–همچنین با نام صفات–نیز شناخته می‌شوند. این اصطلاحات ممکن است به یک نمونه خاص اشاره کنند، مانند «ویژگی (feature) مسافت پیموده شده این ماشین برابر ۱۵۰۰۰ است» یا به کل داده‌ها، مثلاً «ویژگی (feature) مسافت پیموده شده رابطه قوی‌ای با قیمت دارد».

تفاوت‌های کلیدی Feature و Attribute

- Feature بیشتر در زمینه یادگیری ماشین و مدل‌سازی استفاده می‌شود. مثلاً در یک مدل پیش‌بینی قیمت خانه، ویژگی‌ها ممکن است شامل متراژ، تعداد اتاق‌ها، موقعیت مکانی و غیره باشند.

- Attribute معمولاً در زمینه پایگاه داده‌ها یا توصیف اشیاء به کار می‌رود. مثلاً در یک جدول پایگاه داده، ستون‌ها را attributes می‌نامند.

- در عمل، هر attribute می‌تواند تبدیل به یک feature شود، اما ممکن است نیاز به پیش‌پردازش داشته باشد. مثلاً attribute «رنگ» ممکن است به featureهای عددی مثل one-hot encoding تبدیل شود.

یک مثال کاربردی

فرض کنیم داده‌ای درباره خودروها داریم:

© 2025 Mohsen Zare. All rights reserved.

Attribute	مقدار نمونه
رنگ	قرمز
قدرت موتور	۱۵۰ اسب بخار
نوع سوخت	بنزین

در یادگیری ماشین، این‌ها ممکن است به صورت زیر تبدیل شوند:

- رنگ ← featureهای باینری (قرمز = ۱، آبی = ۰، ...)
- قدرت موتور ← feature عددی (۱۵۰)
- نوع سوخت ← feature دسته‌ای (بنزین = ۰، دیزل = ۱، ...)

در کتاب‌ها و منابع مختلف، این دو واژه گاهی به جای هم استفاده می‌شوند. اما اگر بخواهیم دقیق باشیم:

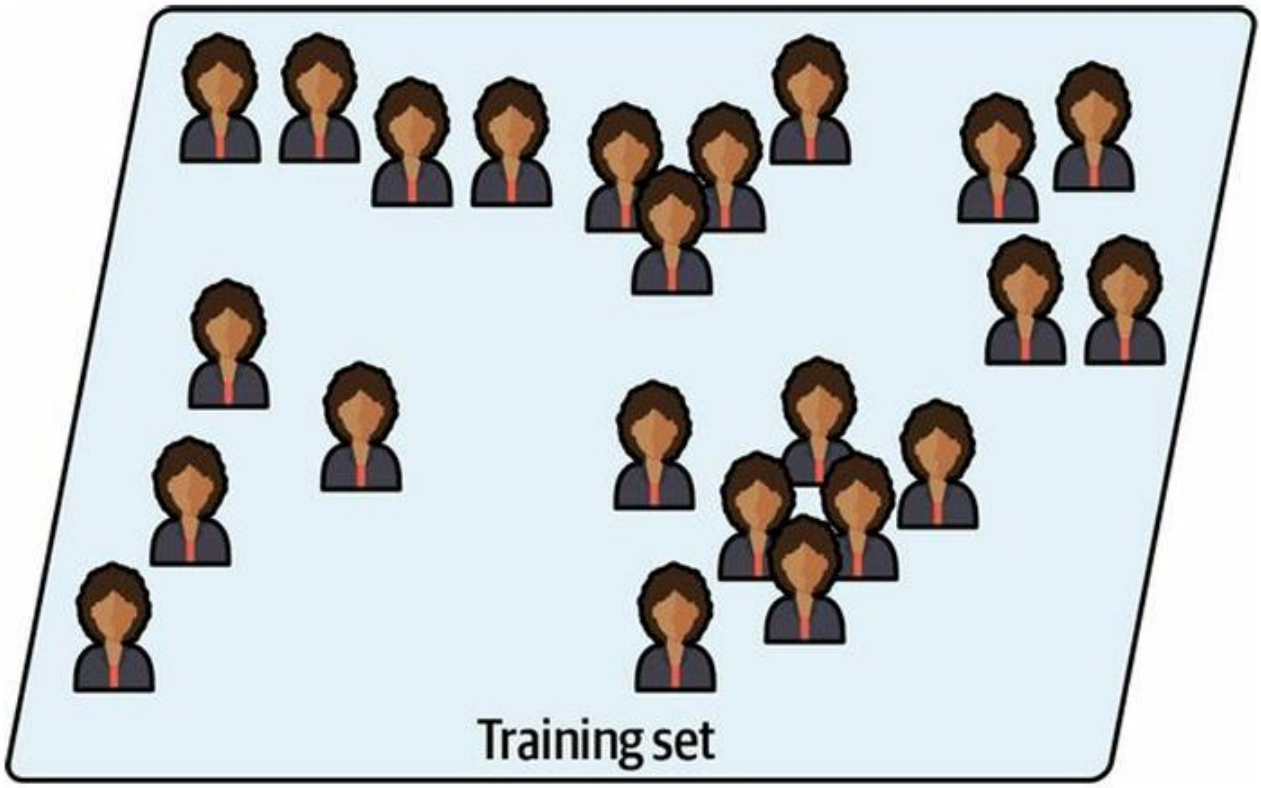
- Attribute: توصیف اولیه داده‌ها، معمولاً در مرحله جمع‌آوری یا ذخیره‌سازی.
- Feature: داده‌های پردازش‌شده و آماده برای استفاده در مدل یادگیری ماشین.

اگر بخوایم این تفاوت رو در ترجمه این کتاب هم رعایت کنیم، بهتره feature رو «ویژگی مدل» یا «ویژگی یادگیری» ترجمه کنیم، و attribute رو «مشخصه داده» یا «ویژگی توصیفی».

© 2025 Mohsen Zare. All rights reserved.

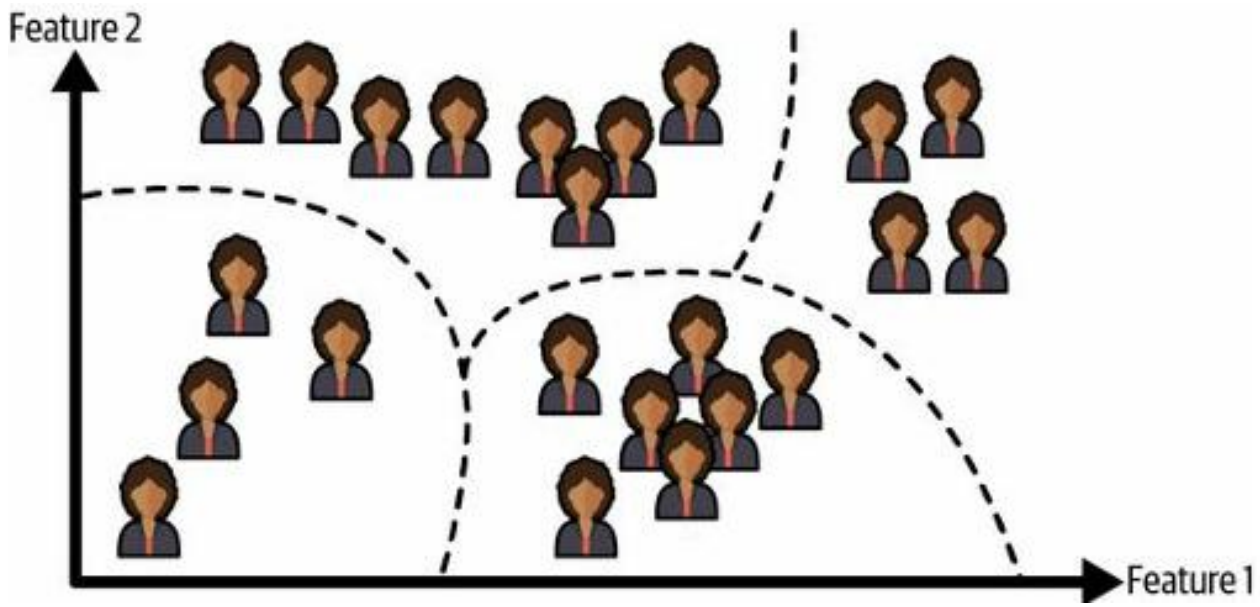
یادگیری بدون نظارت

در «یادگیری بدون نظارت(*unsupervised learning*)»، همانطور که از نامش پیداست، داده‌های آموزشی برچسب‌گذاری نشده‌اند (شکل ۱-۷). در این حالت، سیستم بدون راهنمایی مستقیم سعی می‌کند الگوها و ساختارهای نهفته در داده‌ها را کشف کند. برای مثال، فرض کنید مجموعه‌ای از داده‌های مربوط به بازدیدکنندگان وبلاگ خود دارید. ممکن است بخواهید از یک الگوریتم «خوشه‌بندی(*clustering*)» استفاده کنید تا گروه‌های مشابه بازدیدکنندگان را شناسایی کنید (شکل ۱-۸). در این فرآیند، هیچگاه به الگوریتم نمی‌گویید که هر بازدیدکننده به کدام گروه تعلق دارد؛ بلکه الگوریتم خودش این ارتباط‌ها را کشف می‌کند. مثلاً ممکن است تشخیص دهد که ۴۰% از بازدیدکنندگان شما نوجوانانی هستند که به کتاب‌های کَمیک علاقه‌مندند و معمولاً بعد از مدرسه وبلاگ را مطالعه می‌کنند، در حالی که ۲۰% بزرگسالانی هستند که عاشق داستان‌های علمی تخیلی‌اند و بیشتر در تعطیلات آخر هفته به وبلاگ سر می‌زنند. اگر از الگوریتم «خوشه‌بندی سلسله‌مراتبی (*hierarchical clustering*)» استفاده کنید، حتی ممکن است هر گروه را به زیرگروه‌های کوچک‌تر تقسیم کند. این نوع تحلیل به شما کمک می‌کند تا محتوا و پست‌های وبلاگ‌تان را متناسب با هر گروه هدف‌گذاری کنید و بازده بهتری داشته باشید.



شکل ۱-۷. یک مجموعه آموزشی بدون برچسب برای یادگیری بدون نظارت

© 2025 Mohsen Zare. All rights reserved.

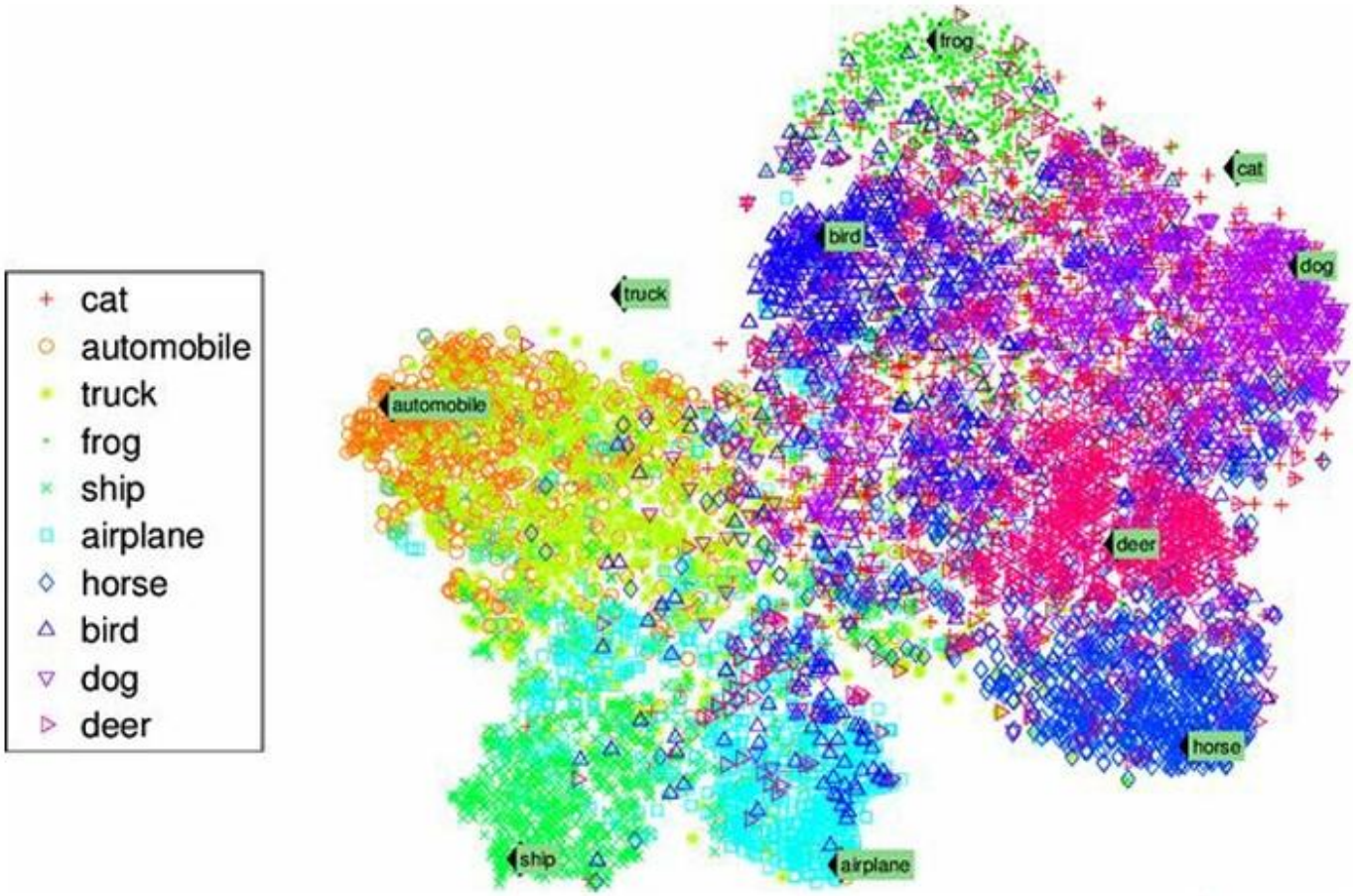


شکل ۱-۸. خوشه‌بندی

الگوریتم‌های «مُصوَرسازی (*Visualization*)»، نمونه‌ای برجسته از یادگیری بدون نظارت هستند. در این روش، حجم زیادی از داده‌های پیچیده و بدون برچسب به الگوریتم داده می‌شود و خروجی آن، نَمایی دوبُعدی یا سه‌بُعدی از داده‌ها است که به راحتی قابل ترسیم و بررسی است (شکل ۱-۹). این الگوریتم‌ها تلاش می‌کنند ساختار اصلی داده‌ها را حفظ کنند؛ به‌عنوان مثال، سعی دارند از همپوشانی خوشه‌های جداگانه در فضای ورودی در فرآیند مصورسازی جلوگیری کنند تا شما بتوانید نحوه سازماندهی داده‌ها را بهتر درک کرده و شاید الگوهای پنهان و ناشناخته‌ای را کشف کنید.

یکی از روش‌های مرتبط با مصورسازی، «کاهش ابعاد (*dimensionality reduction*)» است که هدف آن ساده‌سازی داده‌ها بدون از دست دادن اطلاعات مهم است. یکی از راه‌های انجام این کار، ترکیب چند ویژگی که به هم وابسته‌اند، در یک ویژگی جدید است. برای مثال، مسافت پیموده شده یک خودرو ممکن است رابطه نزدیکی با سن آن داشته باشد؛ بنابراین الگوریتم کاهش ابعاد این دو ویژگی را در یک ویژگی واحد ادغام می‌کند که نمایانگر میزان فرسودگی خودرو است. این فرایند را «استخراج ویژگی (*feature extraction*)» می‌نامند.

© 2025 Mohsen Zare. All rights reserved.



شکل ۱-۹. نمونه‌ای از مصورسازی با استفاده از الگوریتم t-SNE را نشان می‌دهد که خوشه‌های معنایی مختلف را به‌وضوح برجسته می‌کند. این روش داده‌های پیچیده چندبُعدی را به فضایی دو یا سه‌بُعدی تبدیل می‌کند تا ساختار و گروه‌بندی‌های درونی داده‌ها بهتر قابل مشاهده و تحلیل باشد.

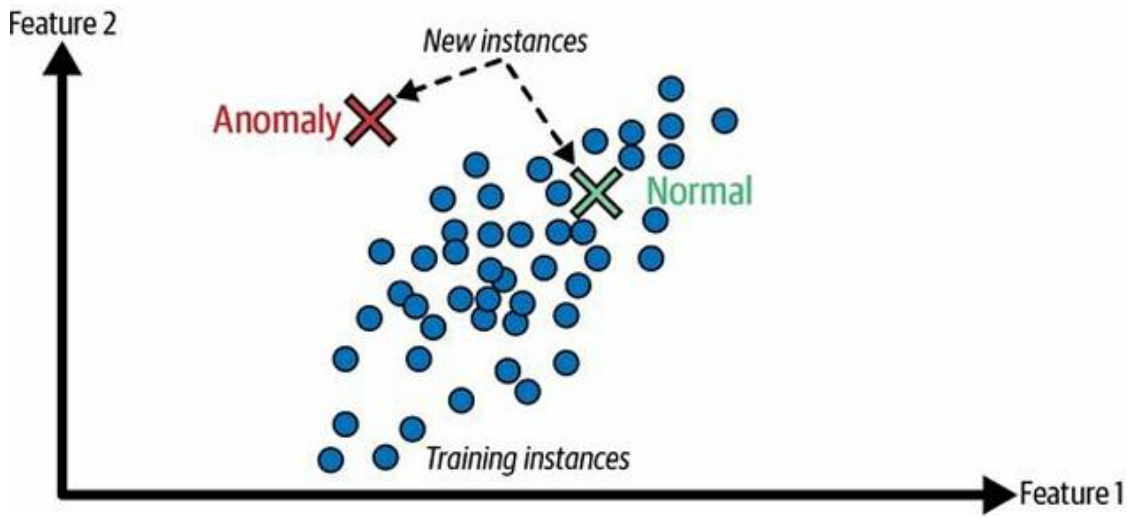
نکته

اغلب بهتر است پیش از ارائه داده‌های آموزشی به الگوریتم‌های یادگیری ماشین دیگر، مانند الگوریتم‌های یادگیری نظارت‌شده، ابتدا با استفاده از الگوریتم‌های کاهش ابعاد، تعداد ویژگی‌ها را کاهش دهید. این کار باعث می‌شود پردازش سریع‌تر انجام شود، داده‌ها فضای کمتری در حافظه و دیسک اشغال کنند و در برخی موارد حتی عملکرد مدل بهبود یابد.

یکی دیگر از مسائل مهم در یادگیری بدون نظارت، «تشخیص ناهنجاری (*anomaly detection*)» است. مثلاً شناسایی تراکنش‌های غیرمعمول کارت اعتباری برای جلوگیری از کلاهبرداری، تشخیص نقص‌های تولید یا حذف خودکار داده‌های پَرت از مجموعه داده(دِیتاسِت) قبل از ارسال به الگوریتم

© 2025 Mohsen Zare. All rights reserved.

یادگیری دیگر. در این روش، سیستم در طول آموزش عمدتاً نمونه‌های عادی را می‌بیند و یاد می‌گیرد که آنها را بشناسد. سپس وقتی نمونه جدیدی دریافت می‌کند، می‌تواند تشخیص دهد که آیا این نمونه شبیه نمونه‌های عادی است یا احتمالاً یک ناهنجاری است (شکل ۱-۱۰ را ببینید). مسئله‌ای مشابه، «تشخیص موارد جدید (*novelty detection*)»–شناسایی داده‌های نوظهور–است که هدفش شناسایی نمونه‌هایی است که کاملاً متفاوت از نمونه‌های موجود در مجموعه آموزشی به نظر می‌رسند. برای این کار باید مجموعه آموزشی بسیار «تمیز» باشد و فاقد نمونه‌هایی باشد که می‌خواهید به عنوان موارد جدید شناسایی کنید. برای مثال، اگر هزاران تصویر سگ دارید که ۱% آن‌ها تصاویر چی‌واوا(Chihuahua) هستند، الگوریتم تشخیص موارد جدید نباید تصاویر جدید چی‌واوا را به عنوان نمونه‌های جدید طبقه‌بندی کند. در مقابل، الگوریتم‌های تشخیص ناهنجاری ممکن است همین تصاویر را به دلیل کمیاب بودن و تفاوت زیاد نسبت به سایر سگ‌ها، به عنوان ناهنجاری تشخیص دهند (که البته این توهینی به چی‌واواها نیست!).



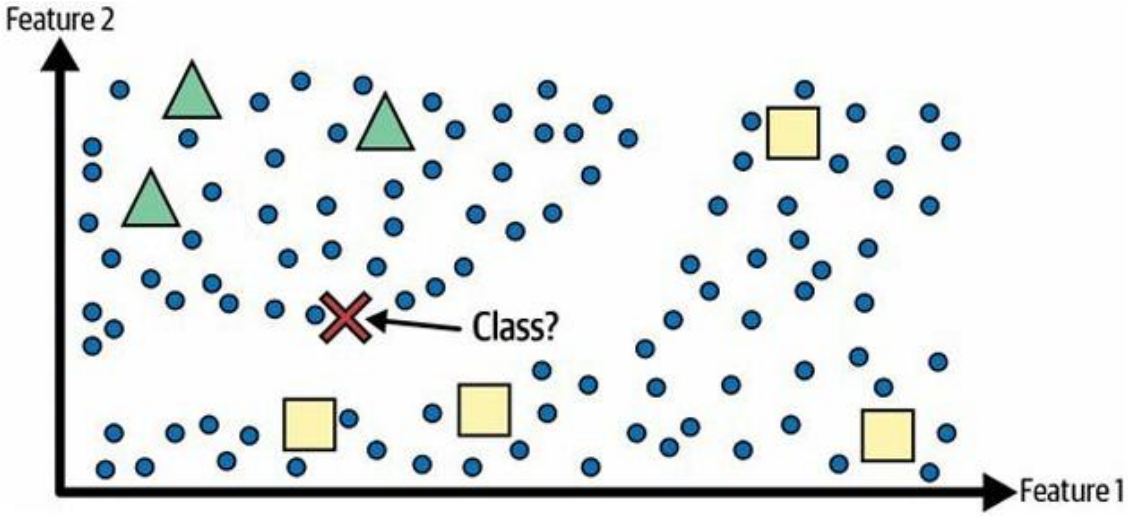
شکل ۱-۱۰. تشخیص ناهنجاری

در نهایت، یکی دیگر از مسائل متداول در یادگیری بدون نظارت، «یادگیری قوانین وابستگی (*association rule learning*)» است. در این روش هدف، کاوش در حجم عظیمی از داده‌ها برای کشف روابط جالب و معنی‌دار بین ویژگی‌ها است. مثلاً فرض کنید صاحب یک سوپرمارکت هستید. اجرای الگوریتم قانون وابستگی روی گزارش‌های فروش ممکن است نشان دهد مشتریانی که سُس باربیکیو و چیپس سیب‌زمینی می‌خرند، اغلب تمایل دارند استیک هم خریداری کنند. در نتیجه، شما ممکن است تصمیم بگیرید این اقلام را در کنار هم قرار دهید تا فروش بهتری داشته باشید.

© 2025 Mohsen Zare. All rights reserved.

یادگیری نیمه‌نظارتی

از آنجا که برچسب‌گذاری داده‌ها معمولاً زمان‌بر و پرهزینه است، اغلب با حجم زیادی از داده‌های بدون برچسب و تعداد کمی نمونه برچسب‌خورده مواجه می‌شوید. برخی الگوریتم‌ها قادرند با این نوع داده‌ها که ترکیبی از نمونه‌های برچسب‌دار و بدون برچسب هستند، به خوبی کار کنند. این رویکرد را «یادگیری نیمه‌نظارتی(*semi-supervised learning*)» می‌نامند (شکل ۱-۱۱).



شکل ۱-۱۱. یادگیری نیمه‌نظارتی با دو کلاس (مثلث و مربع) را نشان می‌دهد. در این مثال، نمونه‌های بدون برچسب (دایره‌ها) به تصمیم‌گیری برای طبقه‌بندی یک نمونه جدید (علامت ضربدر) کمک می‌کنند؛ به طوری که حتی اگر این نمونه به مربع‌های برچسب‌خورده نزدیک‌تر باشد، سیستم آن را در کلاس مثلث قرار می‌دهد.

برخی از سرویس‌های میزبانی عکس مثل Google Photos، نمونه‌ی خوبی از یادگیری نیمه‌نظارتی هستند. وقتی عکس‌های خانوادگی‌تان را در این سرویس آپلود می‌کنید، الگوریتم خوشه‌بندی (که بخشی از یادگیری بدون نظارت است) به‌طور خودکار تشخیص می‌دهد که شخص A در عکس‌های ۱، ۵ و ۱۱ و شخص دیگری به نام B در عکس‌های ۲، ۵ و ۷ حضور دارد. در این مرحله، تنها کاری که سیستم نیاز دارد این است که شما به آن بگویید هر فرد کیست؛ کافی است برای هر نفر یک برچسب بگذارید. سپس سیستم می‌تواند نام افراد را در همه‌ی عکس‌ها بنویسد که این کار برای جستجوی سریع و دقیق عکس‌ها بسیار کاربردی است.

بیشتر الگوریتم‌های یادگیری نیمه‌نظارتی، ترکیبی از روش‌های یادگیری بدون نظارت و یادگیری نظارت‌شده هستند. برای مثال، ابتدا از یک الگوریتم خوشه‌بندی استفاده می‌شود تا نمونه‌های مشابه را در گروه‌هایی دسته‌بندی کند. سپس، هر نمونه‌ی بدون برچسب در هر خوشه، با رایج‌ترین برچسب

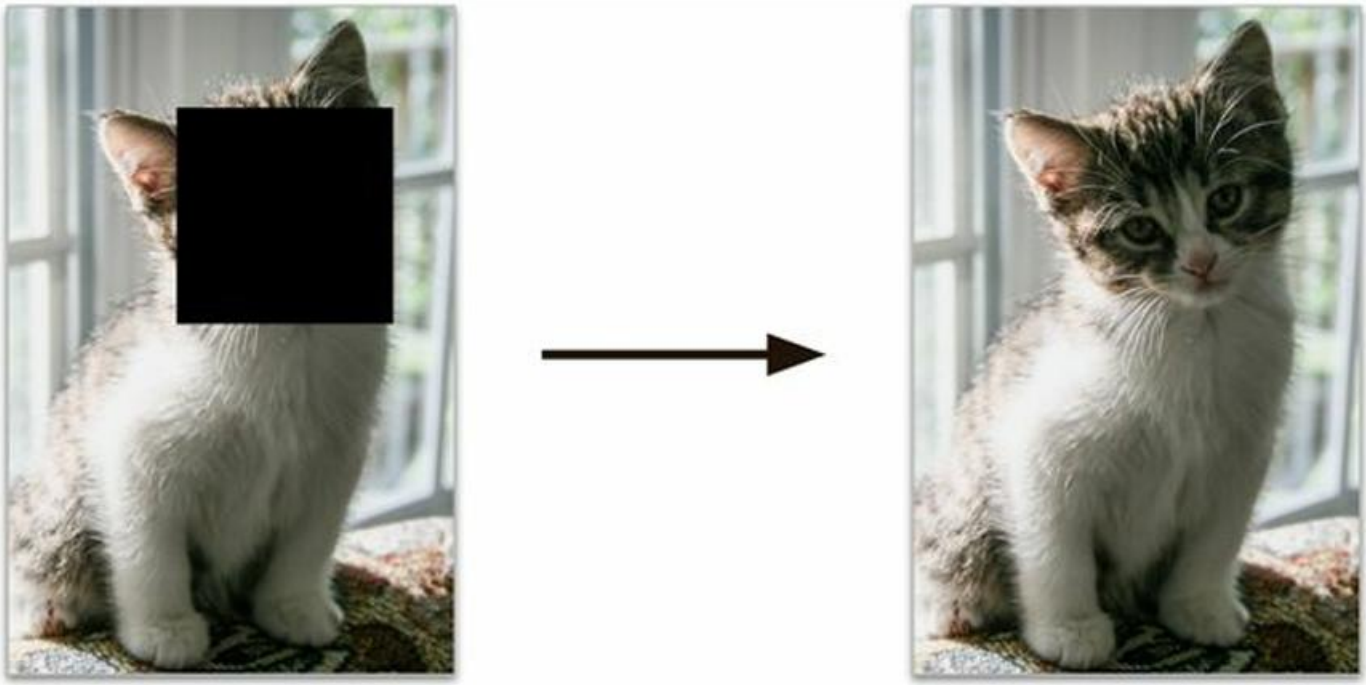
© 2025 Mohsen Zare. All rights reserved.

آن خوشه علامت‌گذاری می‌شود. پس از این مرحله‌ی برچسب‌گذاری کاملِ مجموعه‌داده، می‌توان از هر الگوریتم یادگیری نظارت‌شده برای آموزش مدل استفاده کرد.

یادگیری خود نظارتی

رویکرد دیگری در یادگیری ماشین وجود دارد که در آن، ابتدا یک مجموعه‌داده‌ی کاملاً بدون برچسب را به صورت خودکار برچسب‌گذاری می‌کنیم و سپس با استفاده از این مجموعه‌داده جدید و برچسب‌خورده، مدل‌های یادگیری نظارت‌شده را آموزش می‌دهیم. این روش را «یادگیری خودنظارتی (*self-supervised learning*)» می‌نامند.

برای مثال، فرض کنید مجموعه‌داده‌ی بزرگی از تصاویر بدون برچسب دارید. در این روش، به طور تصادفی بخشی از هر تصویر را ماسک می‌کنید و سپس مدل را طوری آموزش می‌دهید که تصویر اصلی را بازسازی کند(شکل ۱-۱۲). در این فرآیند، تصاویر ماسک‌شده به عنوان ورودی به مدل داده می‌شوند و تصاویر اصلی به عنوان برچسب‌ها عمل می‌کنند.



شکل ۱-۱۲. مثالی از یادگیری خودنظارتی: تصویر ورودی (چپ) و تصویر هدف (راست)

© 2025 Mohsen Zare. All rights reserved.

مدل به‌دست‌آمده از این روش ممکن است به تنهایی کاملاً کاربردی باشد؛ برای نمونه، می‌توان از آن برای ترمیم تصاویر آسیب‌دیده یا حذف اشیاءِ ناخواسته از تصاویر استفاده کرد. اما در بسیاری موارد، مدل آموزش‌دیده با یادگیری خودنظارتی، هدف نهایی نیست. معمولاً قصد دارید این مدل را برای انجام یک کار کمی متفاوت–کاری که واقعاً برای شما اهمیت دارد–تنظیم و بهینه‌سازی کنید.

برای مثال، فرض کنید هدف شما ایجاد یک مدل طبقه‌بندی حیوانات خانگی باشد؛ مدلی که با دیدن عکس هر حیوان، بتواند گونه آن را تشخیص دهد. اگر مجموعه‌ی بزرگی از تصاویر بدون برچسب حیوانات خانگی داشته باشید، می‌توانید ابتدا با یادگیری خودنظارتی مدلی برای ترمیم تصاویر آموزش دهید. وقتی این مدل خوب عمل کند، قادر خواهد بود گونه‌های مختلف حیوانات را تشخیص دهد؛ برای نمونه، هنگام ترمیم تصویری از گربه که قسمتی از صورتش پوشانده شده است، می‌داند نباید به جای آن صورت سگ را اضافه کند.

با فرض اینکه معماری مدل شما چنین قابلیتی داشته باشد (که معمولاً اکثر معماری‌های شبکه عصبی دارند)، می‌توانید مدل را طوری تنظیم کنید که به جای ترمیم تصاویر، گونه حیوانات خانگی را پیش‌بینی کند. این مرحله‌ی آخر، شامل تنظیمِ دقیقِ مدل روی یک مجموعه‌داده‌ی برچسب‌گذاری شده است؛ چرا که مدل از قبل با ویژگی‌های گربه‌ها، سگ‌ها و سایر گونه‌ها آشنا شده، و حالا فقط باید ارتباط بین این شناخت و برچسب‌های مورد انتظار را یاد بگیرد.

نکته

انتقال دانش از یک مسئله به مسئله دیگر را «یادگیری انتقالی (*transfer learning*)» می‌نامند. این روش امروزه یکی از مهم‌ترین تکنیک‌ها در یادگیری ماشین به شمار می‌رود، به‌خصوص وقتی با «شبکه‌های عصبی عمیق (*deep neural networks*)» کار می‌کنیم–یعنی شبکه‌هایی که شامل لایه‌های متعددی از نورون‌ها هستند. در ترجمه‌ی بخش دوم کتاب(جلد دوم ترجمه)، به طور مفصل و جامع به این موضوع پرداخته خواهد شد.

برخی افراد یادگیری خودنظارتی را زیرمجموعه‌ای از یادگیری بدون نظارت می‌دانند، زیرا در این روش با داده‌های کاملاً بدون برچسب کار می‌شود. اما از آنجا که یادگیری خودنظارتی در طول آموزش از برچسب‌های تولید شده‌ی خودکار استفاده می‌کند، از این جهت به یادگیری نظارت‌شده نزدیک‌تر است.

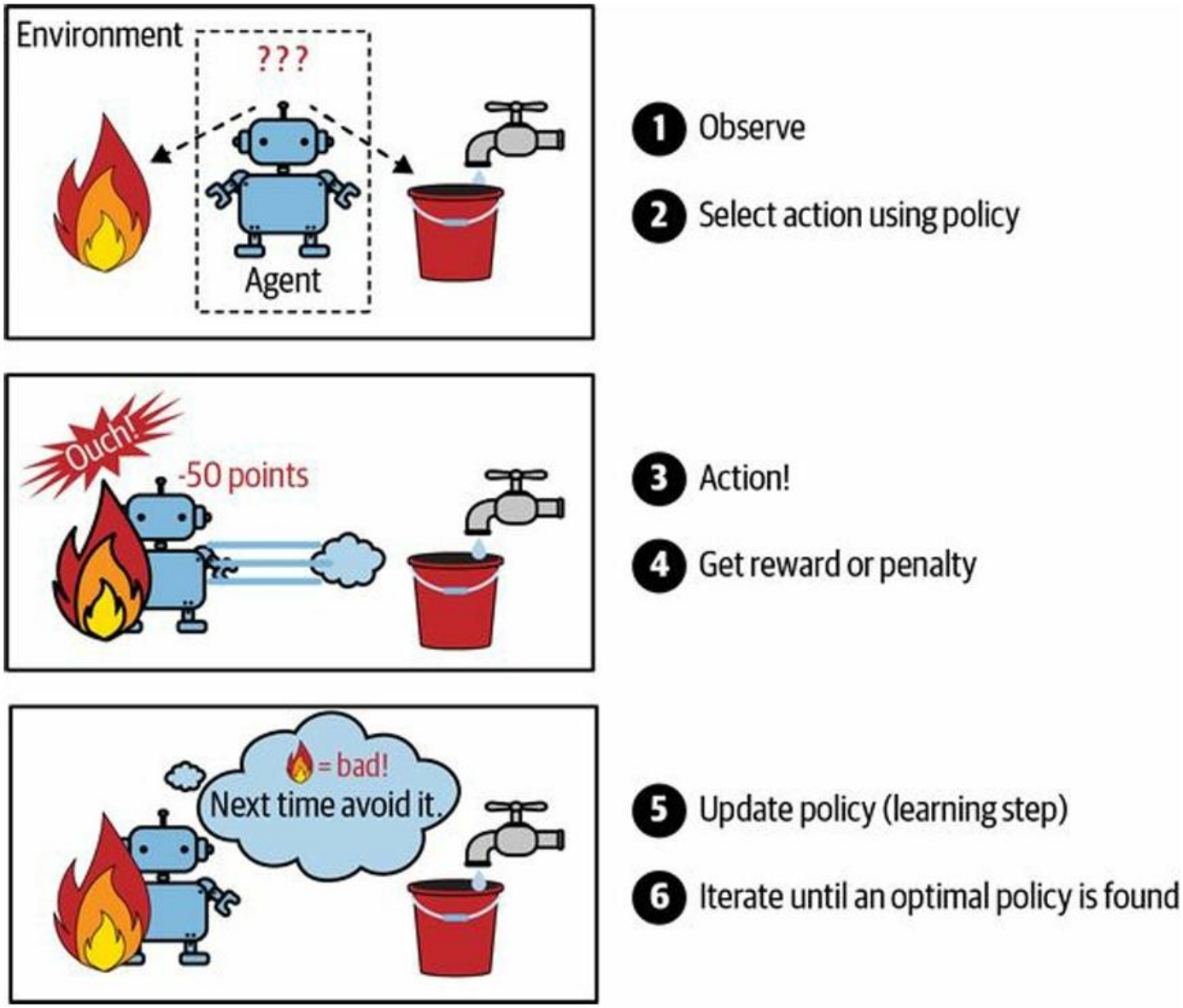
© 2025 Mohsen Zare. All rights reserved.

معمولاً اصطلاح «یادگیری بدون نظارت» بیشتر برای مسائلی مانند خوشه‌بندی، کاهش ابعاد و تشخیص ناهنجاری به کار می‌رود، در حالی که یادگیری خودنظارتی عمدتاً روی همان مسائلی یادگیری نظارت‌شده، یعنی طبقه‌بندی و رگرسیون، تمرکز دارد. به طور خلاصه، بهتر است یادگیری خودنظارتی را به عنوان یک دسته‌بندی مستقل در نظر بگیریم.

یادگیری تقویتی

«یادگیری تقویتی ($Reinforcement\ Learning$)» پدیده‌ای کاملاً متفاوت است. در این روش، سیستمِ یادگیری که به آن «عامل ($agent$)» گفته می‌شود، محیط اطرافش را مشاهده می‌کند، اقداماتی انجام می‌دهد و در پاسخ، «پاداش ($reward$)» دریافت می‌کند (یا ممکن است پاداش‌های منفی به شکل «جریمه ($penalty$)» بگیرد، همانطور که در شکل ۱-۱۳ می‌بینید). هدف $agent$ این است که به مرور زمان، بهترین استراتژی ممکن—که به آن «سیاست ($policy$)» به معنیِ قاعده یا تابع انتخاب عمل گفته می‌شود—را بیاموزد تا بیشترین میزان پاداش را کسب کند. سیاست مشخص می‌کند که عامل در هر موقعیت یا وضعیت خاص، کدام عمل را باید انتخاب کند.

© 2025 Mohsen Zare. All rights reserved.



شکل ۱-۱۳. یادگیری تقویتی

برای نمونه، بسیاری از ربات‌ها از الگوریتم‌های یادگیری تقویتی برای یادگیری مهارت‌هایی مانند راه رفتن استفاده می‌کنند. برنامه AlphaGo شرکت DeepMind نیز مثالی برجسته از این رویکرد است: این برنامه در ماه مه ۲۰۱۷ با شکست دادن Ke Jie، که در آن زمان رتبه یک جهان در بازی Go را داشت، به تیتر یک خبرها تبدیل شد. AlphaGo ابتدا با تحلیل میلیون‌ها بازی و سپس با انجام بازی‌های متعدد در برابر خودش، سیاست برنده شدن را آموخت. جالب است که در بازی‌های واقعی مقابل قهرمان، فرآیند یادگیری فعال نبود؛ بلکه AlphaGo صرفاً سیاستی را که پیش‌تر آموخته بود اجرا می‌کرد. این نوع یادگیری را «یادگیری آفلاین (*offline learning*)» می‌نامند.

© 2025 Mohsen Zare. All rights reserved.

یادگیری دسته‌ای در مقابل یادگیری آنلاین

معیار دیگری برای طبقه‌بندی سیستم‌های یادگیری ماشین، توانایی آن‌ها در یادگیری تدریجی از جریان داده‌های ورودی است. در این حالت، سیستم به جای آموزش یک‌باره روی یک مجموعه‌داده‌ی ثابت، داده‌ها را به‌صورت پیوسته دریافت کرده و دانش خود را به‌روزرسانی می‌کند.

یادگیری دسته‌ای

در «یادگیری دسته‌ای (*batch learning*)»، سیستم توانایی یادگیری تدریجی ندارد و باید یک‌باره با کل مجموعه‌داده‌ی موجود آموزش ببیند. این فرایند معمولاً زمان‌بر و همراه با مصرفِ منابع زیاد است و اغلب در حالت آفلاین انجام می‌شود. در این رویکرد، ابتدا مدل با همه‌ی داده‌ها آموزش داده شده و سپس وارد مرحله‌ی بهره‌برداری می‌شود، جایی که بدون یادگیری جدید فقط آنچه را قبلاً آموخته به کار می‌گیرد. این روش را «یادگیری آفلاین (*offline learning*)» می‌نامند.

متأسفانه عملکرد یک مدل به مرور زمان به‌تدریج کاهش پیدا می‌کند، چون دنیا همیشه در حال تغییر است، در حالی که مدل بدونِ تغییر باقی می‌ماند. این پدیده اغلب «فرسودگی مدل (*model rot*)» یا «تغییر تدریجی توزیع داده (*data drift*)»—تغییر توزیع داده‌های ورودی با زمان—نامیده می‌شود. راه‌حل این مشکل، آموزش مجدد(retrain) مدل به طور منظم با داده‌های به‌روز است. اینکه این کار هر چند وقت یک‌بار لازم است، بستگی به کاربرد مدل دارد: اگر مدل تصاویر گربه و سگ را طبقه‌بندی کند، عملکردش خیلی آهسته اُفت می‌کند، اما اگر مدل با سیستم‌های سریع‌التغییر، مثل پیش‌بینی بازار مالی، سروکار داشته باشد، عملکردش احتمالاً خیلی سریع اُفت می‌کند.

© 2025 Mohsen Zare. All rights reserved.

هشدار

حتی مدلی که برای طبقه‌بندی تصاویر گربه‌ها و سگ‌ها آموزش دیده است نیز ممکن است به بازآموزی منظم نیاز داشته باشد؛ نه به این دلیل که ویژگی‌های زیستی گربه‌ها و سگ‌ها به‌طور ناگهانی تغییر می‌کند، بلکه به این علت که فناوری‌های ثبت تصویر—از جمله دوربین‌ها، فرمت‌های فایل، وضوح، نورپردازی و نسبت ابعاد—همواره در حال تحول هستند. افزون بر این، تغییرات در سلیقه و رفتار انسان نیز می‌تواند تأثیرگذار باشد؛ برای مثال، ممکن است در سال آینده نژادهای دیگری محبوب شوند یا صاحبان حیوانات خانگی تصمیم بگیرند آن‌ها را با پوشش‌ها و لوازم تزئینی جدید بیارایند.

در یک سیستم یادگیری دسته‌ای، اگر بخواهید آن را با داده‌های جدید—مثلاً نوع تازه‌ای از هرزنامه—آشنا کنید، لازم است مدل را از ابتدا و بر پایه کل مجموعه‌داده آموزش دهید؛ یعنی نه‌تنها داده‌های جدید، بلکه داده‌های پیشین را هم دوباره وارد فرایند آموزش کنید. سپس مدل قدیمی با نسخه تازه جایگزین می‌شود. خوشبختانه، همان‌طور که در شکل ۱-۳ دیدید، کل مراحل آموزش، ارزیابی و استقرار یک سیستم یادگیری ماشین را می‌توان به‌سادگی خودکارسازی کرد. بنابراین، حتی یک مدلِ دسته‌ای نیز می‌تواند با تغییرات هماهنگ شود: کافی است داده‌ها را به‌روز کنید و هر زمان که لازم شد، کل مدل را از صفر بازآموزی کنید.

این رویکرد ساده و در بسیاری از موارد کارآمد است، اما آموزش مدل با استفاده از کل مجموعه‌داده‌ها می‌تواند ساعت‌ها زمان ببرد. به همین دلیل، معمولاً بازآموزی سیستم تنها هر ۲۴ ساعت یک‌بار یا حتی به‌صورت هفتگی انجام می‌شود. با این حال، اگر سیستم شما باید با داده‌هایی که به‌سرعت تغییر می‌کنند—مانند پیش‌بینی قیمت سهام—هماهنگ بماند، به رویکردی نیاز دارید که واکنش سریع‌تری داشته باشد.

علاوه بر این، آموزشِ مدل بر روی کل مجموعه‌داده، به منابع محاسباتی قابل‌توجهی نیاز دارد—از جمله پردازنده، حافظه، فضای ذخیره‌سازی، ورودی/خروجی دیسک، و پهنای‌باند شبکه. اگر حجم داده‌ها زیاد باشد و فرآیند آموزش روزانه به‌صورت خودکار انجام شود، هزینه‌های عملیاتی می‌تواند به‌سرعت افزایش یابد. در مواردی که داده‌ها بسیار حجیم باشند، حتی ممکن است استفاده از الگوریتم‌های یادگیری دسته‌ای به‌طور کامل غیرعملی یا غیرممکن شود.

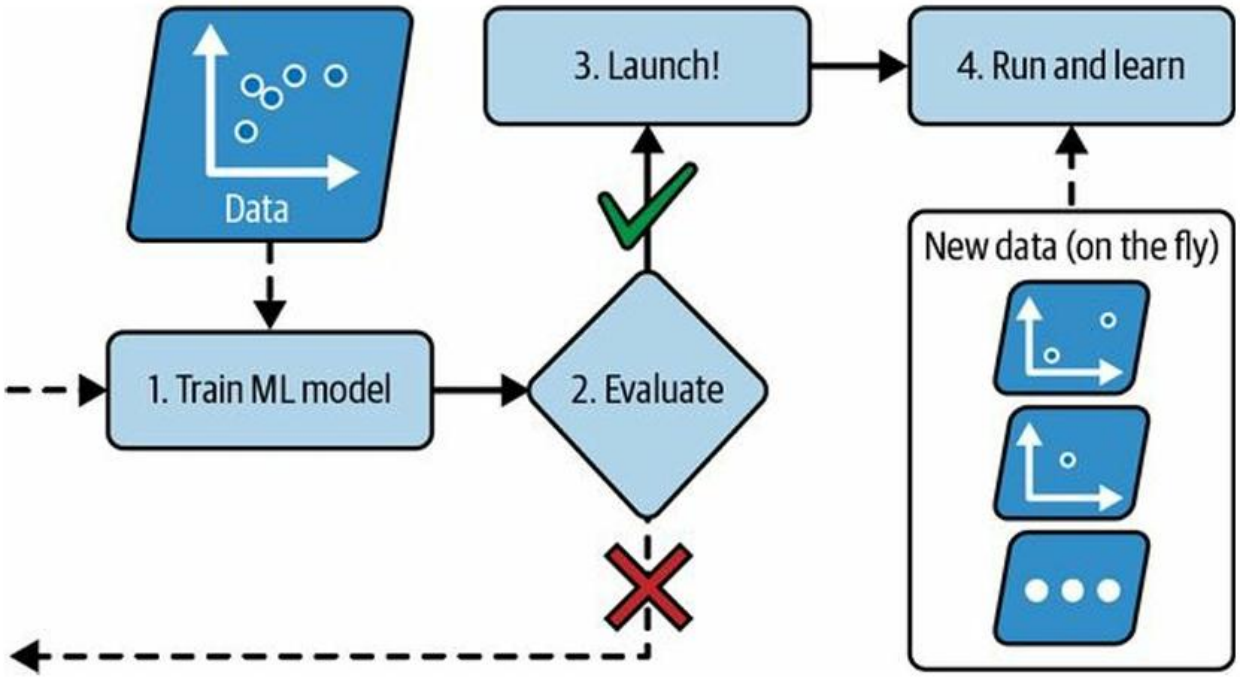
© 2025 Mohsen Zare. All rights reserved.

در نهایت، اگر سیستم شما نیازمند یادگیری خودکار باشد اما با محدودیت‌های منابع مواجه شود—مانند یک اپلیکیشن موبایلی یا یک مریخ‌نورد در حال مأموریت—انتقال حجم زیادی از داده‌های آموزشی و صرف منابع قابل‌توجه برای آموزش چندساعته‌ی روزانه، می‌تواند به‌شدت دشوار و حتی غیرعملی باشد.

گزینه بهتر در همه این موارد، استفاده از الگوریتم‌هایی است که قابلیت یادگیری تدریجی دارند.

یادگیری آنلاین

در «یادگیری آنلاین (*Online Learning*)»، آموزش مدل به‌صورت تدریجی انجام می‌شود؛ بدین شکل که داده‌ها به ترتیبِ ورود، چه به‌صورت تکی و چه در قالب دسته‌ها یا گروه‌های کوچک موسوم به «مینی‌دسته‌ها(*Mini-Batches*)» یا دسته‌های کوچک، به سیستم داده می‌شوند. هر مرحله آموزش سبُک، سریع و کم‌هزینه است، بنابراین مدل می‌تواند بلافاصله پس از دریافت داده‌های جدید، آن‌ها را پردازش کرده و دانش خود را به‌روزرسانی کند (همان‌طور که در شکل ۱-۱۴ نشان داده شده است).

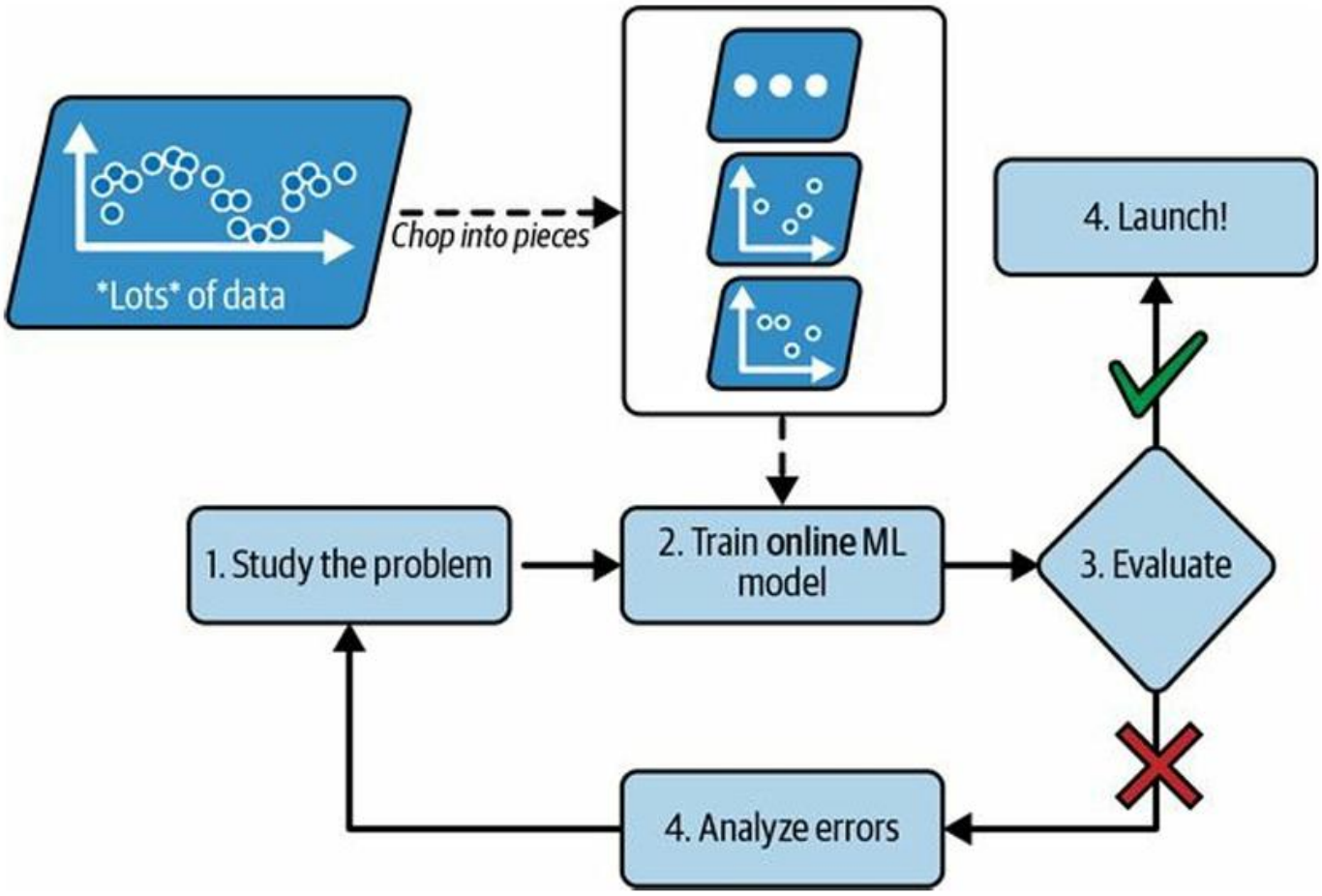


شکل ۱-۱۴. در یادگیری آنلاین، مدل ابتدا آموزش داده می‌شود و وارد مرحله تولید می‌گردد، اما برخلاف یادگیری دسته‌ای، با ورود داده‌های جدید، یادگیری مدل ادامه پیدا می‌کند و به‌تدریج دانش خود را به‌روزرسانی می‌کند.

© 2025 Mohsen Zare. All rights reserved.

یادگیری آنلاین برای سیستم‌هایی که نیاز به سازگاری سریع با تغییرات دارند، مانند تشخیص الگوهای نوظهور در بازار سهام، بسیار مناسب است. این رویکرد همچنین برای محیط‌هایی با منابع محاسباتی محدود، مانند آموزش مدل‌ها روی دستگاه‌های تلفن همراه، گزینه‌ای کارآمد و مقرون‌به‌صرفه محسوب می‌شود.

علاوه بر این، الگوریتم‌های یادگیری آنلاین برای آموزش مدل‌ها روی مجموعه‌داده‌های بسیار بزرگ که در حافظه اصلی یک دستگاه جای نمی‌گیرند، کاربرد دارند؛ به این روش «یادگیری خارج از هسته (*out-of-core learning*)» یا یادگیری بیرون از حافظه اصلی گفته می‌شود. در این رویکرد، الگوریتم بخشی از داده‌ها را بارگذاری کرده و یک مرحله آموزشی روی آن انجام می‌دهد و این فرآیند را تا پوشش تمام داده‌ها تکرار می‌کند (شکل ۱-۱۵).



شکل ۱-۱۵. نمونه‌ای از آموزش آنلاین که اجازه می‌دهد مدل‌ها روی مجموعه‌داده‌های بسیار بزرگ، مرحله‌به‌مرحله و بدون نیاز به بارگذاری تمام داده‌ها در حافظه، آموزش ببینند.

© 2025 Mohsen Zare. All rights reserved.

یکی از پارامترهای کلیدی در سیستم‌های یادگیری آنلاین، سرعت سازگاری با داده‌های در حال تغییر است که به آن «نرخ یادگیری (*learning rate*)» گفته می‌شود. نرخ یادگیریِ بالا باعث می‌شود مدل به سرعت با داده‌های جدید سازگار شود، اما در عین حال داده‌های قدیمی را نیز سریع فراموش می‌کند—برای مثال، یک فیلتر اسپم ممکن است فقط آخرین نوع اسپم را تشخیص دهد و انواع قبلی را نادیده بگیرد. برعکس، نرخ یادگیری پایین باعث می‌شود مدل کندتر یاد بگیرد و نسبت به «نویز داده‌ها یا نقاط پرت (outliers)» کمتر حساس باشد، اما انعطاف‌پذیری آن در برابر تغییرات سریع کمتر خواهد بود.

هشدار

یادگیری خارج از هسته معمولاً به صورت آفلاین انجام می‌شود و روی سیستم زنده اجرا نمی‌شود، بنابراین اصطلاح «یادگیری آنلاین» ممکن است کمی گمراه‌کننده باشد. بهتر است آن را به عنوان «یادگیری افزایشی (*incremental learning*)» یا یادگیری تدریجی در نظر بگیرید.

یک چالش مهم در آموزش آنلاین این است که بسته به کیفیت داده‌ها و نرخ یادگیری، ورود داده‌های نامناسب می‌تواند عملکرد سیستم را به سرعت کاهش دهد. در یک سیستم زنده، این مشکل بلافاصله برای کاربران قابل مشاهده خواهد بود. به عنوان مثال، داده‌های نادرست ممکن است ناشی از یک باگ سخت‌افزاری (مثل حسگر خراب در یک ربات) یا اقدامات عمدی برای فریب سیستم (مانند ارسال هرزنامه به موتور جستجو برای ارتقای رتبه) باشند. برای کاهش این ریسک، باید سیستم را به دقت نظارت کنید و در صورت مشاهده افت عملکرد، فوراً یادگیری آنلاین را متوقف کرده و در صورت نیاز، به وضعیت قبلی بازگردید. همچنین می‌توان داده‌های ورودی را رصد و داده‌های غیرعادی را با استفاده از الگوریتم‌های تشخیص ناهنجاری (به فصل ۹ در جلد دوم ترجمه کتاب مراجعه کنید) مدیریت کرد.

© 2025 Mohsen Zare. All rights reserved.

یادگیری مبتنی بر نمونه در مقابل یادگیری مبتنی بر مدل

یک روش دیگر برای دسته‌بندی سیستم‌های یادگیری ماشین، نحوه «تعمیم (*generalization*)» آنها—توانایی یک مدل برای عملکرد خوب روی داده‌های جدید و نادیده—است. بیشتر مسائل یادگیری ماشین حول محور پیش‌بینی می‌چرخند؛ یعنی سیستم باید بتواند با توجه به نمونه‌های آموزشی، پیش‌بینی‌های دقیقی روی نمونه‌هایی که قبلاً ندیده است انجام دهد. داشتن عملکرد خوب روی داده‌های آموزشی ارزشمند است، اما کافی نیست؛ معیار واقعی موفقیت، توانایی تعمیم به داده‌های جدید است.

دو رویکرد اصلی برای تعمیم وجود دارد: یادگیری مبتنی بر نمونه و یادگیری مبتنی بر مدل.

یادگیری مبتنی بر نمونه

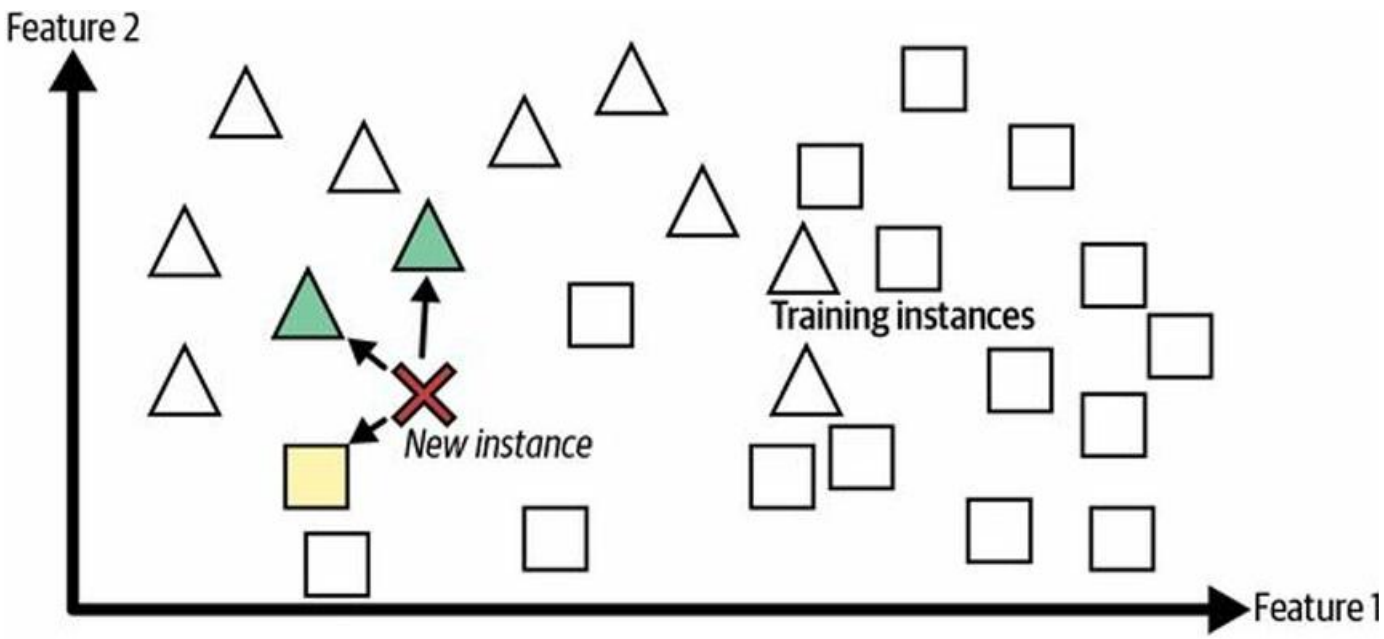
احتمالاً ساده‌ترین شکل یادگیری، «یادگیری از طریق حفظ کردن» است. اگر بخواهید با این روش یک فیلتر اسپم بسازید، سیستم صرفاً ایمیل‌هایی را که مشابه ایمیل‌های قبلی علامت‌گذاری شده‌اند، شناسایی و علامت‌گذاری می‌کند–این روش ممکن است کار کند، اما قطعاً بهترین راه حل نیست.

به جای اینکه فقط ایمیل‌هایی را که دقیقاً با ایمیل‌های اسپم قبلی یکسان هستند علامت‌گذاری کنید، فیلتر اسپم می‌تواند طوری طراحی شود که ایمیل‌هایی با شباهت زیاد به ایمیل‌های اسپم شناخته شده را نیز علامت‌گذاری کند. برای این کار، باید شباهت بین دو ایمیل اندازه‌گیری شود. آن را «معیار شباهت (*measure of similarity*)» می‌نامند. یک معیار ساده اساسی می‌تواند «شمارش کلمات مشترک» بین دو ایمیل باشد؛ سیستم ایمیلی را اسپم می‌داند که تعداد زیادی کلمه مشترک با ایمیل‌های اسپم قبلی داشته باشد.

این روش «یادگیری مبتنی بر نمونه (*instance-based learning*)» نامیده می‌شود: سیستم ابتدا نمونه‌ها را حفظ می‌کند و سپس برای تعمیم به نمونه‌های جدید، آنها را با استفاده از «معیار شباهت» با نمونه‌های آموخته شده (یا زیرمجموعه‌ای از آنها) مقایسه می‌کند. به عنوان مثال، در شکل ۱-۱۶،

© 2025 Mohsen Zare. All rights reserved.

نمونه جدید به عنوان یک مثلث طبقه‌بندی می‌شود، زیرا بیشتر نمونه‌های بسیار شبیه به آن، به این کلاس تعلق دارند.

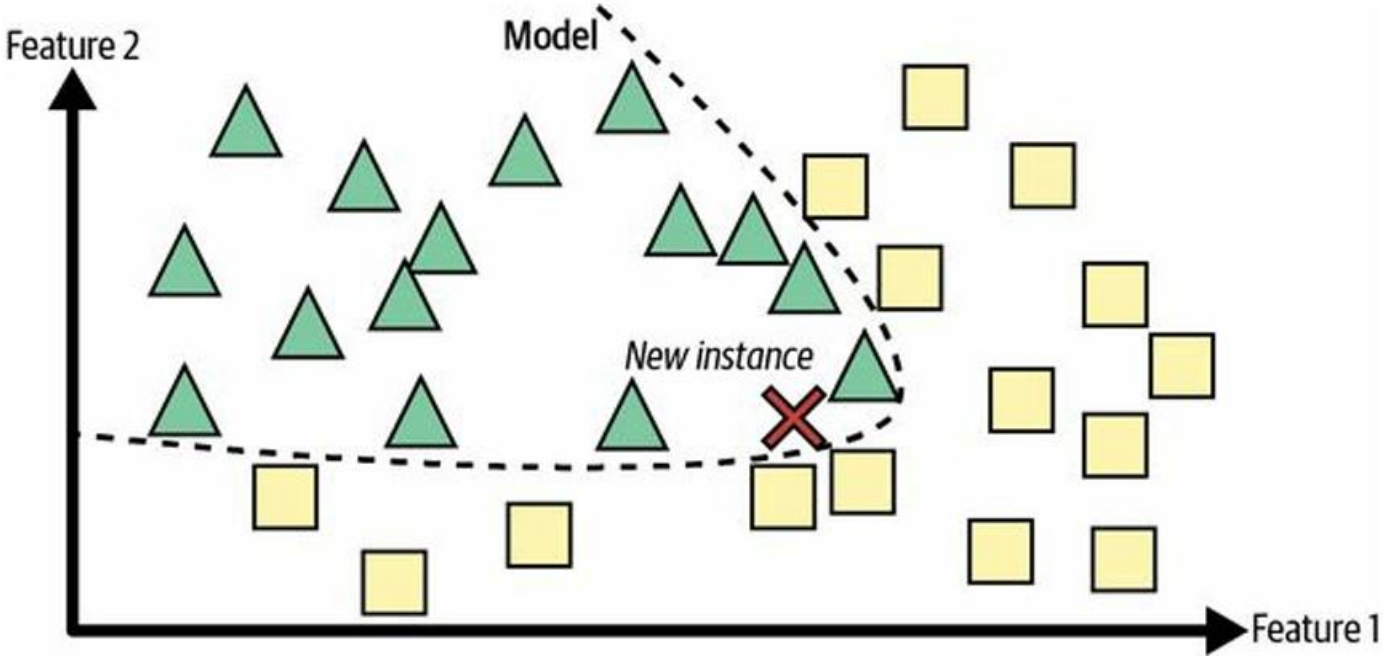


شکل ۱-۱۶. یادگیری مبتنی بر نمونه

یادگیری مبتنی بر مدل و گردش کار معمول یادگیری ماشین

روش دیگر برای تعمیم از روی مجموعه‌ای از مثال‌ها، ساخت یک مدل از این مثال‌ها و سپس استفاده از آن مدل برای انجام «پیش‌بینی‌ها (*predictions*)» است. این رویکرد «یادگیری مبتنی بر مدل (*model-based learning*)» نامیده می‌شود (شکل ۱-۱۷).

© 2025 Mohsen Zare. All rights reserved.



شکل ۱-۱۷. یادگیری مبتنی بر مدل

برای مثال، فرض کنید می‌خواهید رابطه بین پول و شادی مردم را بررسی کنید. در این صورت، داده‌های شاخص زندگی بهتر (Better Life Index) را از وب‌سایت OECD و آمار تولید ناخالص داخلی (GDP) سرانه را از بانک جهانی جمع‌آوری می‌کنید. سپس این جداول را با هم ترکیب کرده و بر اساس مقدار GDP سرانه مرتب می‌کنید. جدول ۱-۱ نمونه‌ای از داده‌های حاصل را نشان می‌دهد.

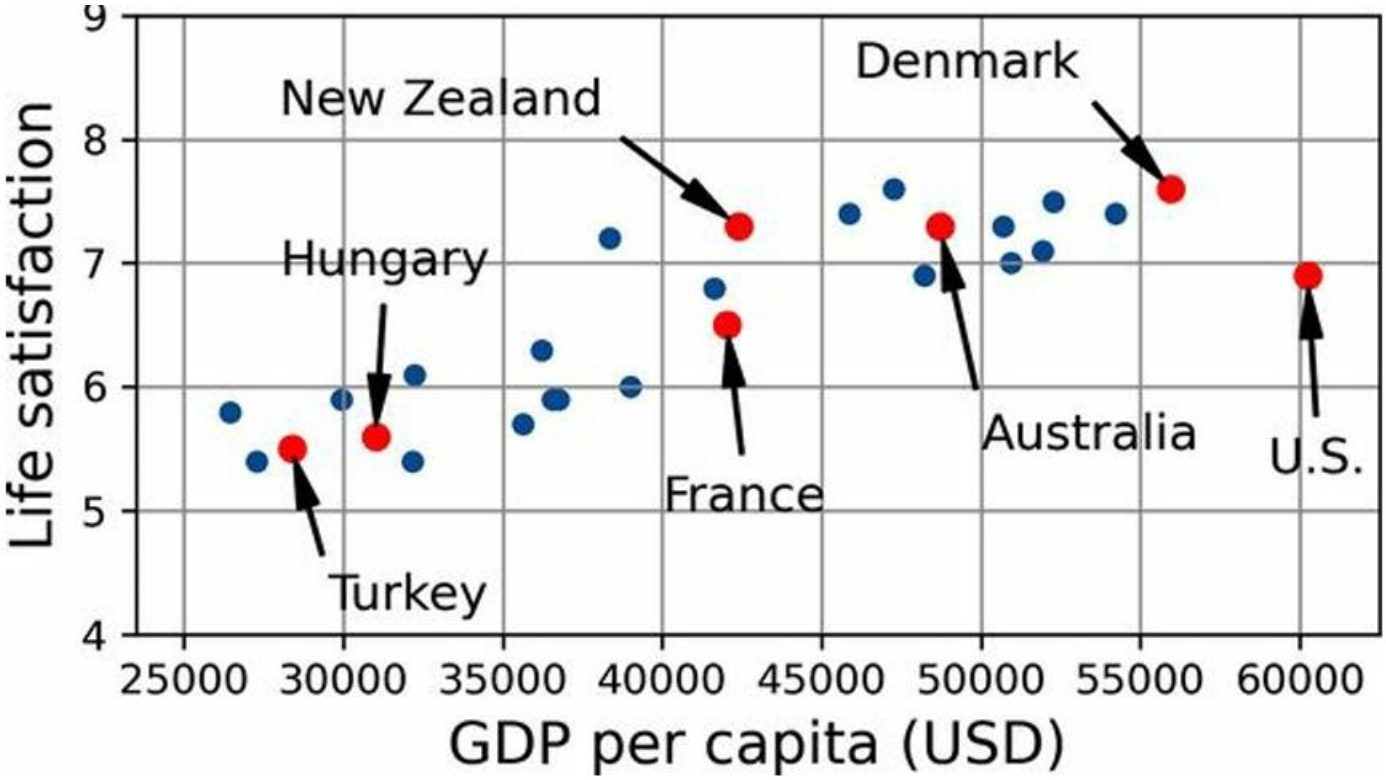
جدول ۱-۱. آیا پول مردم را شادتر می‌کند؟

کشور	تولید ناخالص داخلی سرانه (دلار آمریکا)	رضایت از زندگی
ترکیه	۲۸,۳۸۴	۵/۵
مجارستان	۳۱,۰۰۸	۵/۶
فرانسه	۴۲,۰۲۶	۶/۵
ایالات متحده	۶۰,۲۳۶	۶/۹

© 2025 Mohsen Zare. All rights reserved.

نیوزیلند	۴۲,۴۰۴	۷٫۳
استرالیا	۴۸,۶۹۸	۷٫۳
دانمارک	۵۵,۹۳۸	۷٫۶

حال بیایید داده‌های این کشورها را به تصویر بکشیم (شکل ۱-۱۸).



شکل ۱-۱۸. آیا در اینجا روندی می‌بینید؟

به نظر می‌رسد یک روند در داده‌ها وجود دارد! اگرچه داده‌ها کمی «نویزی (*noisy data*)» هستند، اما رضایت از زندگی با افزایش تولید ناخالص داخلیِ سرانه تقریباً به صورت خطی افزایش می‌یابد. بنابراین تصمیم می‌گیرید رضایت از زندگی را به عنوان یک تابع خطی از تولید ناخالص داخلی سرانه مدل‌سازی کنید. این مرحله «انتخاب مدل (*model selection*)» نامیده می‌شود: شما یک «مدل خطی

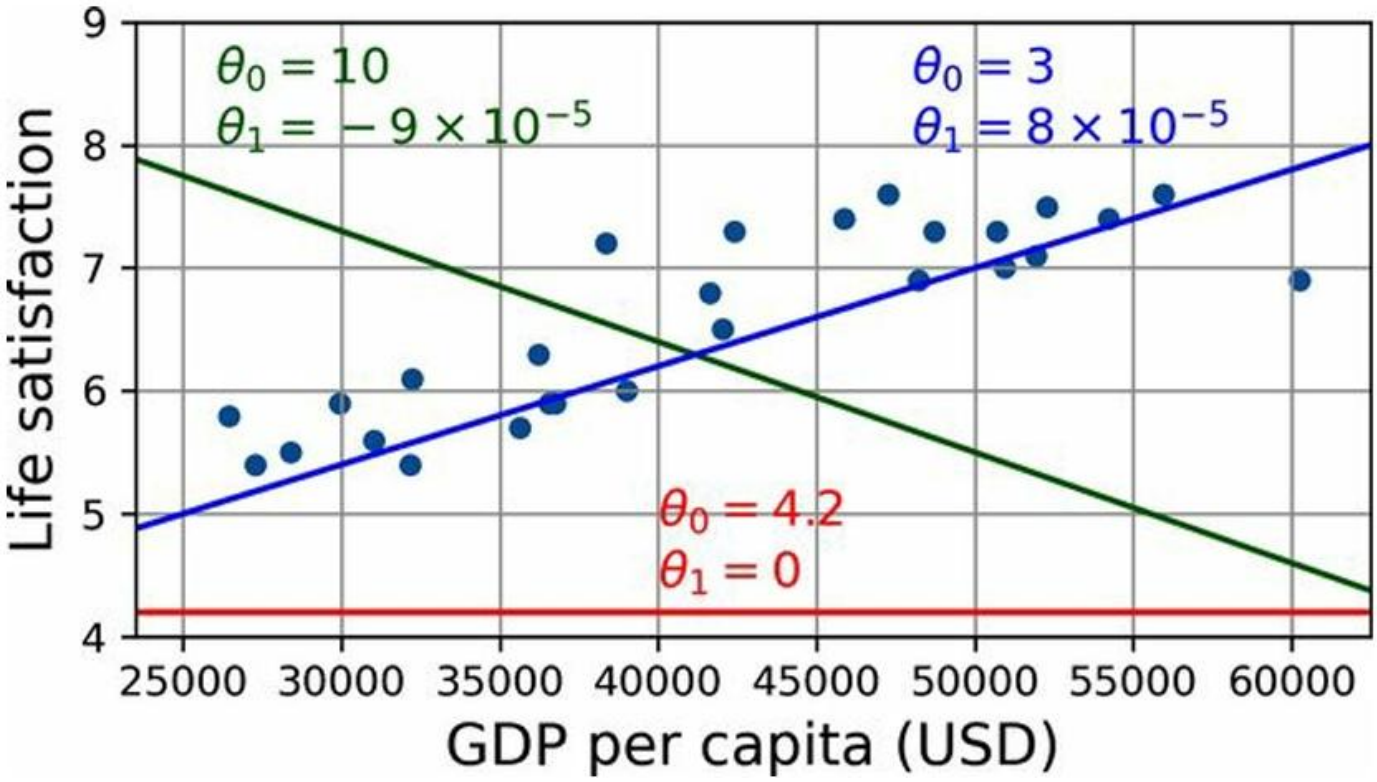
© 2025 Mohsen Zare. All rights reserved.

(*linear model*)» برای رضایت از زندگی با تنها یک ویژگی، یعنی تولید ناخالص داخلی سرانه، انتخاب کرده‌اید (معادله ۱-۱).

معادله ۱-۱. یک مدل خطی ساده

$$\text{life_satisfaction} = \theta_0 + \theta_1 \times \text{GDP_per_capita}$$

این مدل دارای دو «پارامترِ مدل (*model parameters*)» است: θ_0 و θ_1. با تغییر این پارامترها، همان‌طور که در شکل ۱-۱۹ نشان داده شده، می‌توانید مدل خود را طوری تنظیم کنید که هر تابع خطیِ ممکن را نمایش دهد.



شکل ۱-۱۹. نمونه‌هایی از مدل‌های خطی مختلف که با تغییر پارامترهای θ_0 و θ_1 قابل تولید هستند.

قبل از استفاده از مدل، باید مقادیر پارامترهای θ_0 و θ_1 مشخص شوند. پرسش این است: چگونه بفهمیم کدام مقادیر بهترین عملکرد را فراهم می‌کنند؟ برای پاسخ، باید یک «معیار عملکرد (*performance measure*)» تعریف کنید. می‌توان یک «تابع سودمندی (*utility function*)» یا «تابع برازش (*fitness function*)» تعریف کرد که «میزان خوبی» مدل را اندازه‌گیری کند، یا یک «تابع

© 2025 Mohsen Zare. All rights reserved.

هزینه ($cost\ function$)» که «میزان بد بودن» مدل را مشخص سازد. در مسائل رگرسیون خطی، معمولاً از تابع هزینه‌ای استفاده می‌شود که فاصله بین پیش‌بینی‌های مدل و نمونه‌های آموزشی را اندازه‌گیری می‌کند؛ هدف مینیمُم کردن این فاصله برای دستیابی به دقیق‌ترین پیش‌بینی‌ها است.

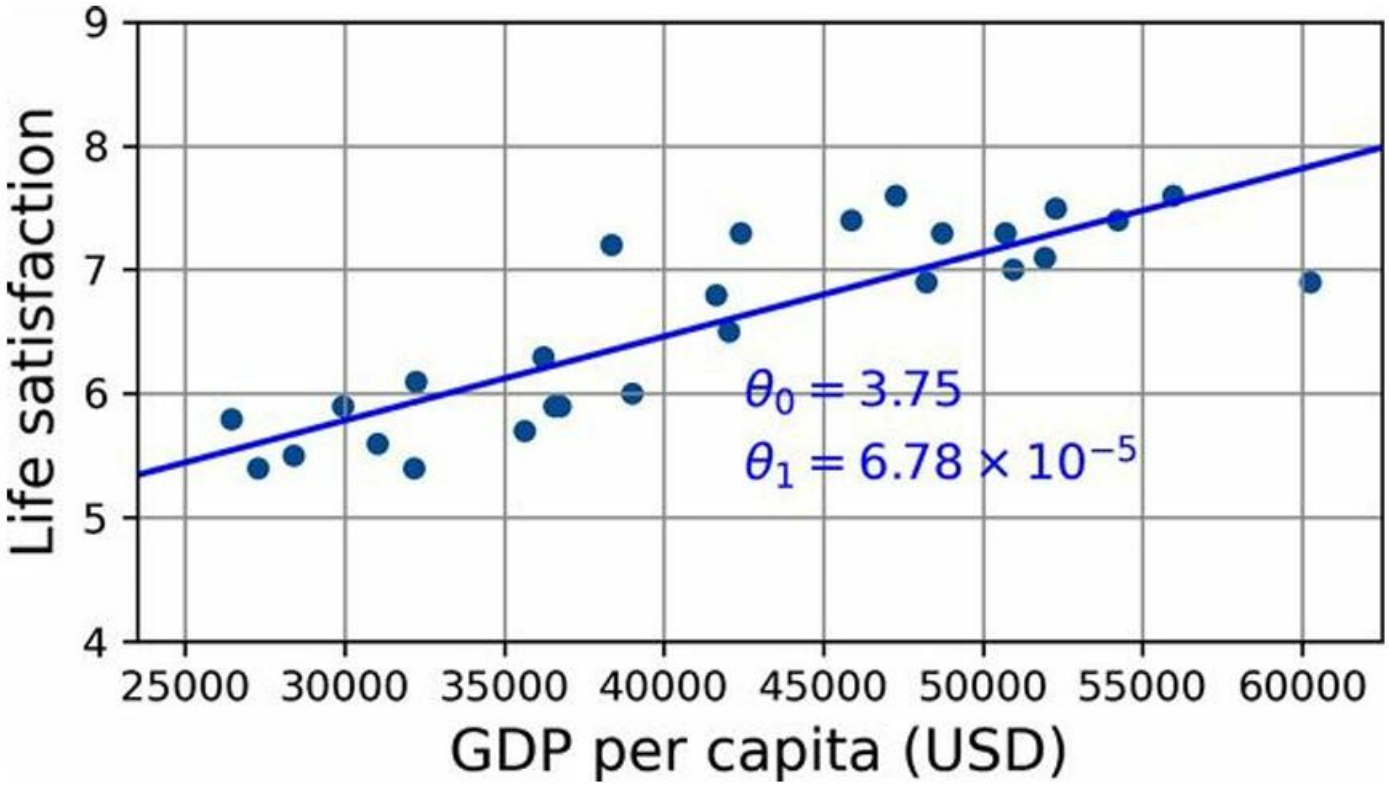
اینجاست که «الگوریتم رگرسیون خطی» وارد عمل می‌شود: شما مثال‌های آموزشی خود را به الگوریتم می‌دهید و الگوریتم پارامترهایی پیدا می‌کند که مدل خطی را به بهترین شکل با داده‌ها برازش (تطبیق) دهند. به این فرایند، «آموزش مدل ($training\ the\ model$)» گفته می‌شود. در مثال مورد بررسی ما، الگوریتم مقادیر بهینه پارامترها را با مقادیرِ $\theta_0 = 3.75$ و $\theta_1 = 6.78 \times 10^{-5}$ پیدا می‌کند.

هشدار

به‌طور گیج‌کننده‌ای، واژه «مدل» می‌تواند به چند معنا اشاره داشته باشد: «نوعی از مدل» (مثلاً رگرسیون خطی)، «یک معماری مشخص از مدل» (مثلاً رگرسیون خطی با یک ورودی و یک خروجی) یا «مدل آموزش‌دیده نهایی» که آماده پیش‌بینی است (مثلاً رگرسیون خطی با همان ورودی و خروجی، با پارامترهای $\theta_0 = 3.75$ و $\theta_1 = 6.78 \times 10^{-5}$). انتخاب مدل شامل تعیین نوع مدل و مشخص کردن کامل معماری آن است. آموزش مدل به معنای اجرای الگوریتمی برای یافتن پارامترهایی است که مدل را به بهترین شکل با داده‌های آموزشی برازش دهند و امیدواریم روی داده‌های جدید نیز پیش‌بینی‌های دقیقی ارائه دهد.

اکنون مدل، تا حد امکان برای یک مدل خطی، با داده‌های آموزشی برازش می‌یابد، همان‌طور که در شکل ۲۰-۱ نشان داده شده است.

© 2025 Mohsen Zare. All rights reserved.



شکل ۱-۲۰. مدل خطی که به بهترین شکل با داده‌های آموزشی برازش دارد

حالا شما آماده‌اید تا مدل خود را برای پیش‌بینی به کار ببرید. برای مثال، اگر بخواهید بدانید مردم قبرس چقدر احساس خوشحالی دارند و داده‌های OECD پاسخی ارائه نمی‌دهند، می‌توانید از مدل خود استفاده کنید: ابتدا سرانه تولید ناخالص داخلی قبرس را بررسی می‌کنید که ۳۷,۶۵۵ دلار است، سپس مدل را اعمال می‌کنید و پیش‌بینی می‌کنید که رضایت از زندگی تقریباً برابر با $3.75 + 37{,}655 \times 6.78 \times 10^{-5} = 6.30$ خواهد بود.

برای روشن شدن موضوع، مثال ۱-۱ کد پایتونی را نشان می‌دهد که ابتدا داده‌ها را بارگذاری کرده، سپس ورودی‌ها (X) را از برچسب‌ها (y) جدا می‌کند، یک نمودار پراکندگی برای تجسم داده‌ها ایجاد می‌کند، و در ادامه یک مدل خطی را آموزش داده و پیش‌بینی‌ها را انجام می‌دهد.

مثال ۱-۱. آموزش و اجرای یک مدل خطی با استفاده از Scikit-Learn

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

© 2025 Mohsen Zare. All rights reserved.

```python
from sklearn.linear_model import LinearRegression

# Download and prepare the data
data_root = "https://github.com/ageron/data/raw/main/"
lifesat = pd.read_csv(data_root + "lifesat/lifesat.csv")
X = lifesat[["GDP per capita (USD)"]].values
y = lifesat[["Life satisfaction"]].values

# Visualize the data
lifesat.plot(kind='scatter', grid=True,
        x="GDP per capita (USD)", y="Life satisfaction")
plt.axis([23_500, 62_500, 4, 9])
plt.show()

# Select a linear model
model = LinearRegression()

# Train the model
model.fit(X, y)

# Make a prediction for Cyprus
X_new = [[37_655.2]]  # Cyprus' GDP per capita in 2020
print(model.predict(X_new)) # output: [[6.30165767]]
```

© 2025 Mohsen Zare. All rights reserved.

توجه

اگر به جای استفاده از مدل خطی، یک الگوریتم یادگیری مبتنی بر نمونه به کار می‌بردید، متوجه می‌شدید که اسرائیل نزدیک‌ترین سرانه تولید ناخالص داخلی به قبرس را دارد (۳۸,۳۴۱ دلار) و با توجه به داده‌های OECD که نشان می‌دهد رضایت از زندگی اسرائیلی‌ها ۷٫۲ است، می‌توانستید پیش‌بینی کنید که رضایت از زندگی قبرس نیز ۷٫۲ باشد. اگر کمی دامنه بررسی را گسترش دهید و دو کشور نزدیکِ بعدی، یعنی لیتوانی و اسلوونی، را در نظر بگیرید که هر دو رضایت از زندگی ۵٫۹ دارند، با میانگین‌گیری این سه مقدار، پیش‌بینی ۶٫۳۳ به دست می‌آید که بسیار نزدیک به پیش‌بینی مدل خطی شماست. این الگوریتم ساده، «رگرسیون k-نزدیک‌ترین همسایه (*k-nearest neighbors regression*)» نامیده می‌شود که در این مثال k=3 است.

برای استفاده از رگرسیون k-نزدیک‌ترین همسایه به جای رگرسیون خطی در کد قبلی، کافی است خطوط مربوط به تعریف و آموزش مدل خطی را با خطوط مربوط به مدل k-نزدیک‌ترین همسایه جایگزین کنید:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

با این دو خط:

```
from sklearn.neighbors import KNeighborsRegressor

model = KNeighborsRegressor(n_neighbors=3)
```

بقیه کد (پیش‌بینی، رسم نمودار و غیره) بدون تغییر باقی می‌ماند.

اگر همه چیز به خوبی پیش برود، مدل شما قادر خواهد بود پیش‌بینی‌های مناسبی ارائه دهد. در غیر این صورت، ممکن است نیاز باشد ویژگی‌های بیشتری وارد کنید (مانند نرخ اشتغال، وضعیت سلامت، آلودگی هوا و غیره)، داده‌ی آموزشی بزرگ‌تر یا با کیفیت‌تر جمع‌آوری کنید، یا حتی یک مدل قوی‌تر، مثلاً رگرسیون چندجمله‌ای، انتخاب نمایید.

© 2025 Mohsen Zare. All rights reserved.

به طور خلاصه:

- ابتدا داده‌ها را بررسی و مطالعه کردید.
- سپس یک مدل مناسب انتخاب کردید.
- مدل را روی داده‌های آموزشی آموزش دادید؛ یعنی الگوریتم یادگیری، مقادیر پارامترهایی را یافت که تابع هزینه را به کمترین حد می‌رسانند.
- در نهایت، از مدل آموزش‌دیده برای پیش‌بینی یا تصمیم‌گیری روی داده‌های جدید استفاده کردید (که به آن «استنتاج (*inference*)» می‌گویند)، با این امید که مدل بتواند به خوبی تعمیم یابد.

این نمونه‌ای از یک پروژه معمولی یادگیری ماشین است. در فصل ۲، شما این فرآیند را به‌طور کامل و عملی تجربه خواهید کرد.

تا اینجا مطالب زیادی را پوشش داده‌ایم: اکنون می‌دانید یادگیری ماشین دقیقاً چیست، چرا کاربردی است، رایج‌ترین دسته‌های سیستم‌های یادگیری ماشین کدامند و یک «گردش کار پروژه (*project workflow*)» معمولی چگونه انجام می‌شود. حالا بیایید نگاهی به مشکلاتی بیندازیم که می‌توانند باعث خطا در یادگیری شوند و مانع پیش‌بینی‌های دقیق شما شوند.

© 2025 Mohsen Zare. All rights reserved.

چالش‌های اصلی یادگیری ماشین

به طور خلاصه، از آنجایی که کار اصلی شما انتخاب یک مدل و آموزش آن بر اساس داده‌ها است، دو عامل اصلی که می‌توانند باعث خطا شوند عبارت‌اند از: «مدل نامناسب» و «داده‌های نامطلوب». بیایید ابتدا با بررسی نمونه‌هایی از داده‌های نامناسب شروع کنیم.

© 2025 Mohsen Zare. All rights reserved.

تعداد داده‌های ناکافی برای آموزش

برای آموزش یک کودک نوپا به شناخت سیب، کافی است یک سیب را نشان دهید و بگویید «سیب»—و احتمالاً این کار را چند بار تکرار کنید. پس از این تمرین ساده، کودک می‌تواند سیب‌ها را در رنگ‌ها و شکل‌های مختلف تشخیص دهد. واقعاً شگفت‌انگیز است!

یادگیری ماشین هنوز به سطح توانایی یک کودک در یادگیری سریع نرسیده است. بیشتر الگوریتم‌های یادگیری ماشین برای عملکرد صحیح، نیازمند حجم زیادی از داده هستند. حتی در مسائل نسبتاً ساده، معمولاً به هزاران نمونه نیاز است و برای مسائل پیچیده‌تر مانند تشخیص تصویر یا گفتار، ممکن است میلیون‌ها نمونه لازم باشد—مگر اینکه بتوانید از بخش‌هایی از یک مدل موجود مجدداً استفاده کنید.

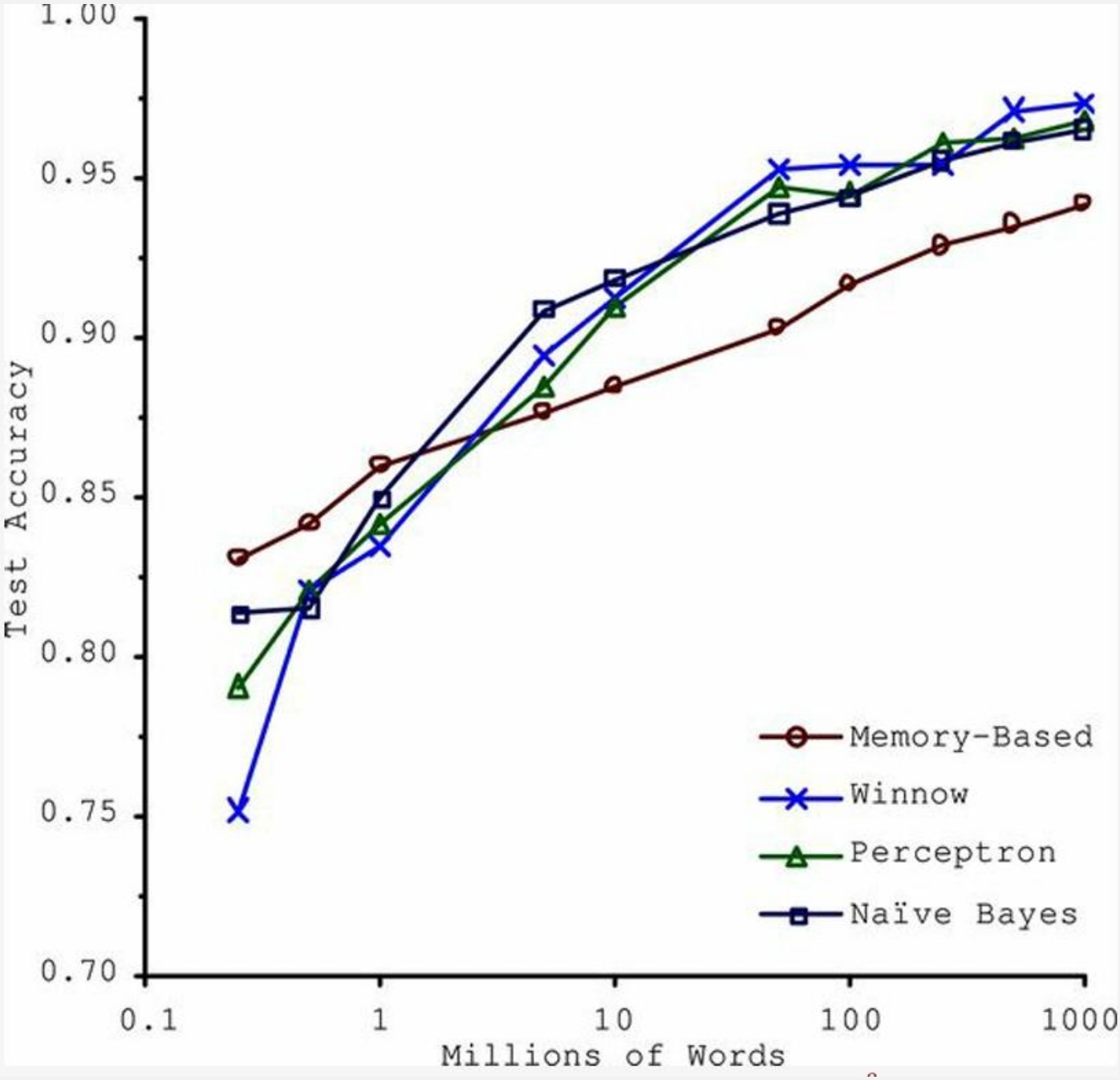
اثربخشی نامعقول داده‌ها

در مقاله‌ای مشهور منتشرشده در سال ۲۰۰۱، محققان مایکروسافت، میشل بنکو و اریک بریل نشان دادند که الگوریتم‌های مختلف یادگیری ماشین—حتی الگوریتم‌های نسبتاً ساده—پس از دریافت حجم کافی از داده‌ها، در حل یک مسئله پیچیده‌ی ابهام‌زداییِ زبانِ طبیعی، تقریباً به همان میزان عملکرد خوبی دارند (شکل ۱-۲۱).

همان‌طور که نویسندگان اشاره می‌کنند، «این نتایج نشان می‌دهد که شاید ارزش آن را داشته باشد که در موردِ توازن (trade-off) بین صرف زمان و هزینه برای توسعه الگوریتم و صرف آن برای توسعه مجموعه داده بازنگری کنیم».

این ایده که داده‌ها در مسائل پیچیده اهمیت بیشتری نسبت به الگوریتم‌ها دارند، توسط پیتر نورویگ و همکارانش در مقاله‌ای با عنوان «اثربخشی نامعقول داده‌ها» در سال ۲۰۰۹ بیشتر ترویج شد. با این حال، باید در نظر داشت که مجموعه‌داده‌های کوچک و متوسط هنوز بسیار رایج هستند و دستیابی به داده‌های آموزشی بیشتر همیشه آسان یا کم‌هزینه نیست—پس هنوز هم نمی‌توان اهمیت الگوریتم‌ها را نادیده گرفت.

© 2025 Mohsen Zare. All rights reserved.



شکل ۱-۲۱. اهمیت داده‌ها در مقابل الگوریتم‌ها

© 2025 Mohsen Zare. All rights reserved.

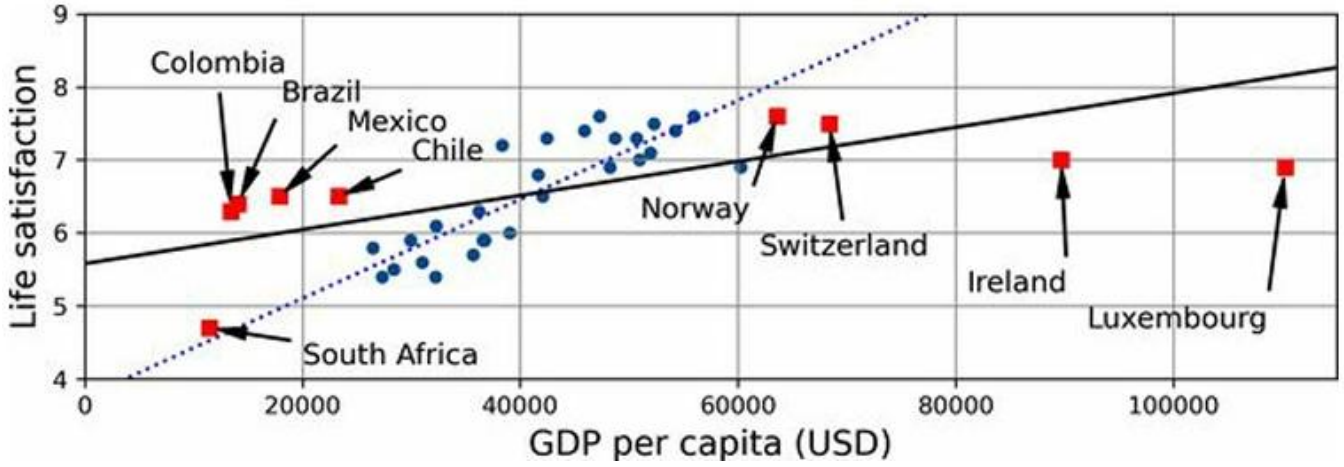
داده‌های آموزشی غیرنماینده

برای دستیابی به تعمیم‌دهی مؤثر، اهمیت زیادی دارد که داده‌های آموزشی نماینده (representative) برای نمونه‌های جدیدی باشند که قصد دارید مدل را روی آن‌ها تعمیم دهید. این اصل، هم در یادگیری مبتنی بر نمونه و هم در یادگیری مبتنی بر مدل برقرار است.

برای مثال، داده‌هایی که قبلاً برای آموزش مدل خطی استفاده شدند، نماینده‌ی کامل نبودند؛ این مجموعه شامل هیچ کشوری با سرانه تولید ناخالص داخلی کمتر از ۲۳,۵۰۰ دلار یا بالاتر از ۶۲,۵۰۰ دلار نبود. شکل ۱-۲۲ نشان می‌دهد که با اضافه کردن این کشورها، داده‌ها چگونه تغییر می‌کنند.

اگر مدل خطی را روی این داده‌های جدید آموزش دهید، خط مُمتَد حاصل می‌شود، در حالی که مدل قبلی با خط نقطه‌چین نشان داده شده است. همان‌طور که مشاهده می‌کنید، اضافه کردن چند کشور جدید نه تنها مدل را به شکل قابل توجهی تغییر می‌دهد، بلکه نشان می‌دهد که یک مدل خطی ساده احتمالاً هرگز عملکرد دقیقی نخواهد داشت. به نظر می‌رسد کشورهای بسیار ثروتمند نسبت به کشورهای نسبتاً ثروتمند شادتر نیستند (در واقع کمی کمتر راضی به نظر می‌رسند)، و برخی از کشورهای فقیر نیز نسبت به بسیاری از کشورهای ثروتمند شادترند.

استفاده از یک مجموعه داده‌ی آموزشیِ غیرنماینده باعث شده مدلی آموزش دهید که احتمالاً پیش‌بینی‌های دقیقی ارائه نخواهد داد، به ویژه برای کشورهای بسیار فقیر و بسیار ثروتمند.



شکل ۱-۲۲. یک نمونه آموزشی نماینده تر

© 2025 Mohsen Zare. All rights reserved.

اطمینان از اینکه مجموعه آموزشی نماینده‌ی مواردی است که قصد دارید مدل به آن‌ها تعمیم داده‌شود، اهمیت زیادی دارد. این کار اغلب دشوارتر از آنچه تصور می‌کنید است: اگر نمونه‌ها خیلی کم باشند، با «نویز نمونه‌گیری (*sampling noise*)» مواجه می‌شوید، یعنی داده‌ها ممکن است به‌صورت تصادفی غیرنماینده باشند. حتی نمونه‌های بزرگ هم در صورت نقص در روش انتخاب داده‌ها، می‌توانند نماینده نباشند؛ به این مشکل «بایاس نمونه‌گیری (*sampling bias*)» گفته می‌شود.

مثال هایی از بایاس نمونه‌گیری

یکی از مشهورترین نمونه‌های بایاس نمونه‌گیری در انتخابات ریاست جمهوری ایالات متحده در سال ۱۹۳۶ رخ داد، زمانی که لاندن با روزولت رقابت می‌کرد. نشریه *Literary Digest* یک نظرسنجی بسیار گسترده برگزار کرد و برای حدود ۱۰ میلیون نفر نامه فرستاد. از این تعداد، ۲/۴ میلیون نفر پاسخ دادند و پیش‌بینی شد که لاندن با ۵۷% آرا برنده خواهد شد. در واقعیت، روزولت با ۶۲% آرا پیروز شد. دلیل این خطا، نقص در روش نمونه‌گیری نشریه *Literary Digest* بود:

- نخست، برای تهیه فهرست آدرس‌های مورد نظر، نشریه *Literary Digest* از دفترچه‌های تلفن، فهرست مشترکین مجلات، فهرست اعضای باشگاه‌ها و منابع مشابه استفاده کرد. این منابع عمدتاً شامل افراد ثروتمندتر بودند که احتمال بیشتری داشت به جمهوری‌خواهان (و به‌ویژه لاندن) رأی دهند.

- دوم، کمتر از ۲۵% از افرادی که دعوت به شرکت در نظرسنجی شدند، پاسخ دادند. این امر موجب ایجاد بایاس نمونه‌گیری شد، زیرا احتمالاً افرادی که به سیاست علاقه نداشتند، از مجله *Literary Digest* خوششان نمی‌آمد یا به دلایل دیگر مشارکت نمی‌کردند، در مطالعه وارد نشده بودند. این نوع بایاس نمونه‌گیری به «بایاس عدم پاسخ (*non-response bias*)» معروف است.

یک مثال دیگر این است: فرض کنید می‌خواهید سیستمی بسازید که موزیک ویدیوهای سبک فانک (funk) را تشخیص دهد. یک روش برای ایجاد مجموعه آموزشی، جستجوی «موسیقی فانک» در یوتیوب (YouTube) و استفاده از ویدیوهای حاصل است. اما این روش مبتنی بر فرض نادرستی

© 2025 Mohsen Zare. All rights reserved.

است: موتور جستجو ویدیوهایی را بازمی‌گرداند که نماینده کل موزیک ویدیوهای فانک در یوتیوب باشند. در واقع، نتایج جستجو معمولاً به سمت هنرمندان محبوب متمایل(دارای بایاس) هستند و اگر در برزیل باشید، ویدیوهای «فانک کاریوکا» زیادی دریافت می‌کنید که شباهتی به جیمز براون ندارند. علاوه بر این، تهیه یک مجموعه آموزشی بزرگ و جامع کار آسانی نیست.

© 2025 Mohsen Zare. All rights reserved.

داده‌های بی‌کیفیت

بدیهی است که اگر داده‌های آموزشی شما پر از خطا(error)، داده‌های پَرت یا نویز(noise) باشد—برای مثال به دلیل اندازه‌گیری‌های بی‌کیفیت—سیستم شما در تشخیص الگوهای اصلی با مشکل مواجه می‌شود و احتمال عملکرد مطلوب آن کاهش می‌یابد. به همین دلیل، صرف زمان برای پاکسازی داده‌های آموزشی معمولاً ارزشمند است. در واقع، بسیاری از دانشمندان داده بخش قابل توجهی از زمان خود را به همین کار اختصاص می‌دهند. چند نمونه از موقعیت‌هایی که نیاز به پاکسازی داده دارند عبارت‌اند از:

- اگر برخی داده‌ها به‌وضوح نقاط پرت هستند، بهتر است آنها را حذف کرده یا خطاهایشان را به‌صورت دستی اصلاح کنید.
- اگر بعضی از نمونه‌ها چند ویژگی را ازدست داده اند (مثلاً ۵% از مشتریان سن خود را ثبت نکرده‌اند)، باید تصمیم بگیرید که آیا این ویژگی را به‌طور کامل نادیده بگیرید، آن نمونه‌ها را کنار بگذارید، مقادیر مفقود را جایگزین کنید (مثلاً با میانگین سن)، یا دو مدل جداگانه—یکی با آن ویژگی و یکی بدون آن—آموزش دهید.

© 2025 Mohsen Zare. All rights reserved.

ویژگی‌های نامرتبط

همان‌طور که می‌گویند: «ورودی بی‌ارزش، خروجی بی‌ارزش». سیستم شما فقط زمانی می‌تواند به‌خوبی یاد بگیرد که داده‌های آموزشی شامل تعداد کافی ویژگی‌های مرتبط باشد و در عین حال، تعداد زیادی از «ویژگی‌های نامرتبط (Irrelevant Features)»—متغیرها یا فیچرهایی که تأثیر مستقیمی بر پیش‌بینی مدل ندارند و فقط باعث افزایش نویز یا پیچیدگی می‌شوند—در آن وجود نداشته باشد. یکی از عوامل کلیدی موفقیت در یک پروژه یادگیری ماشین، فراهم کردن مجموعه‌ای مناسب از ویژگی‌ها برای آموزش است. این کار که «مهندسی ویژگی (*feature engineering*)» نامیده می‌شود، شامل مراحل زیر است:

- «انتخاب ویژگی (*feature selection*)»: گزینش مرتبط‌ترین و مفیدترین ویژگی‌ها برای آموزش از میان ویژگی‌های موجود.
- «استخراج ویژگی (*feature extraction*)»: ترکیب ویژگی‌های موجود برای ساخت یک ویژگی مؤثرتر. (همان‌طور که پیش‌تر دیدیم، الگوریتم‌های کاهش ابعاد می‌توانند در این مرحله کمک کنند.)
- ایجاد ویژگی‌های جدید از طریق جمع‌آوری داده‌های تازه.

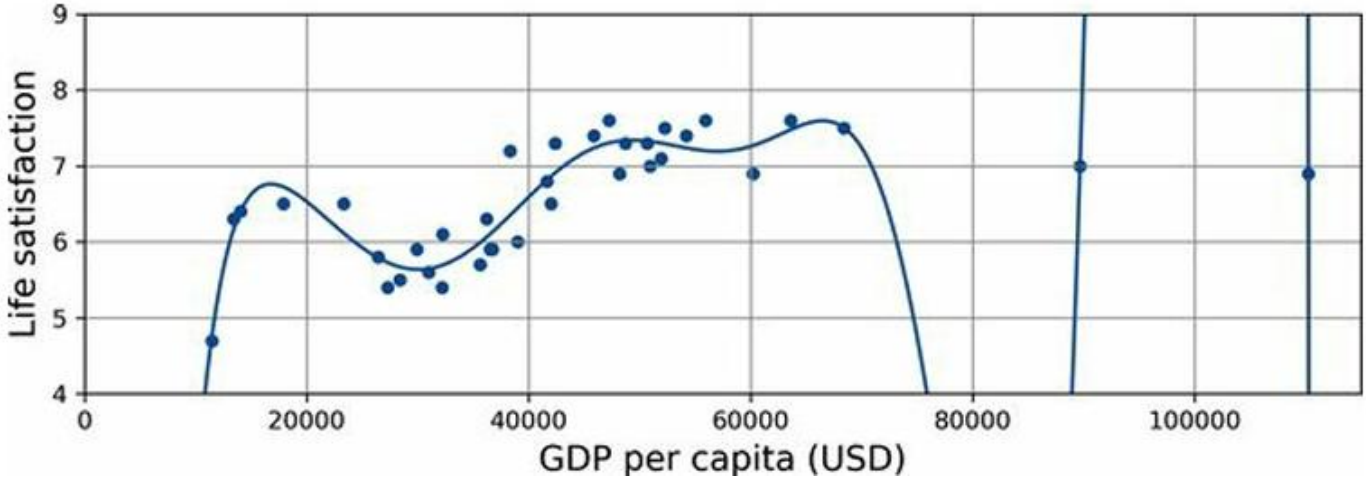
حالا که نمونه‌های متعددی از داده‌های نامناسب را بررسی کردیم، بیایید به چند نمونه از الگوریتم‌های نامناسب هم نگاهی بیندازیم.

© 2025 Mohsen Zare. All rights reserved.

بیش‌برازش داده‌های آموزشی

فرض کنید به یک کشور خارجی سفر کرده‌اید و راننده تاکسی، کرایه‌ای بسیار بیشتر از حد معمول از شما دریافت می‌کند. ممکن است به اشتباه نتیجه بگیرید که «همه‌ی» رانندگان تاکسی در آن کشور دزد هستند. این نمونه‌ای از «تعمیم بیش از حد» است؛ رفتاری که ما انسان‌ها گاهی انجام می‌دهیم و اگر مراقب نباشیم، مدل‌های یادگیری ماشین نیز می‌توانند دچار همین خطا شوند. در حوزه یادگیری ماشین، به این پدیده «بیش‌برازش (*overfitting*)»—منطبق شدن بیش از حد—گفته می‌شود. بیش‌برازش زمانی رخ می‌دهد که مدل روی داده‌های آموزشی عملکرد بسیار خوبی دارد، اما توانایی تعمیم به داده‌های جدید و نادیده را از دست می‌دهد. این پدیده زمانی رخ می‌دهد که مدل به جای یادگیری الگوهای کلی، جزئیات و نویز داده‌های آموزشی را یاد می‌گیرد. نتیجه آن دقت بالا روی داده‌های آموزش و عملکرد ضعیف روی داده‌های جدید است.

شکل ۱-۲۳ نمونه‌ای از یک مدل چندجمله‌ای (polynomial) با درجه بالا برای پیش‌بینی رضایت از زندگی را نشان می‌دهد که به‌شدت دچار بیش‌برازش شده است. هرچند این مدل روی داده‌های آموزشی عملکرد بسیار بهتری نسبت به یک مدل خطی ساده دارد، اما آیا می‌توان واقعاً به پیش‌بینی‌های آن برای داده‌های جدید اعتماد کرد؟



شکل ۱-۲۳. بیش‌برازش داده‌های آموزشی

مدل‌های پیچیده مانند «شبکه‌های عصبی عمیق» توانایی شناسایی الگوهای بسیار ظریف در داده‌ها را دارند، اما اگر مجموعه‌ی آموزشی کوچک یا دارای نویز باشد، خطر شناسایی الگوهای کاذب بالا می‌رود.

© 2025 Mohsen Zare. All rights reserved.

در چنین شرایطی، مدل ممکن است به جای الگوهای واقعی، الگوهای ناشی از نویز را یاد بگیرد؛ مشابه مثال راننده تاکسی.

به‌عنوان نمونه، اگر به مدل رضایت از زندگی خود ویژگی‌های زیادی بدهید، حتی ویژگی‌های نامرتبطی مانند نام کشور، ممکن است مدل «الگوهایی» عجیب کشف کند. مثلاً متوجه شود که در داده‌های آموزشی، تمام کشورهایی که نامشان حرف «w» دارد، رضایت از زندگی بالای ۷ دارند: نیوزیلند (۷٫۳)، نروژ (۷٫۶)، سوئد (۷٫۳) و سوئیس (۷٫۵). اما آیا می‌توان مطمئن بود این «قانون w» برای کشورهایی مثل رواندا(Rwanda) یا زیمبابوه(Zimbabwe) هم صدق می‌کند؟ واضح است که این الگو صرفاً یک اتفاق تصادفی در داده‌هاست، اما مدل ابزاری برای تشخیص واقعی یا جعلی بودن آن ندارد.

هشدار

بیش‌برازش زمانی رخ می‌دهد که پیچیدگی مدل با حجم داده و میزان نویزِ موجود در داده‌های آموزشی متناسب نباشد. راهکارهای اصلی برای مقابله با آن عبارت‌اند از:

- ساده‌سازی مدل: انتخاب مدلی با پارامترهای کمتر (مثلاً مدل خطی به جای مدل چندجمله‌ایِ مرتبه بالا)، کاهش تعداد ویژگی‌ها یا اعمال محدودیت بر مدل.
- افزایش حجم داده‌های آموزشی: جمع‌آوری داده‌های بیشتر برای پوشش بهترِ تنوع و کاهش اثر نویز نمونه‌گیری.
- کاهش نویزِ داده‌ها: برطرف کردن خطاهای موجود، حذف داده‌های پرت و بهبود کیفیت اندازه‌گیری‌ها.

محدود کردن یک مدل برای ساده‌تر کردن آن و کاهش خطر بیش‌برازش را «منظم‌سازی ($Regularization$)» می‌نامند. به عنوان مثال، مدل خطی معرفی‌شده‌ی پیش‌تر، دو پارامتر θ_0 و θ_1 دارد که به الگوریتم دو «درجه آزادی ($degrees\ of\ freedom$)» می‌دهد: تغییر ارتفاع خط (θ_0) و تغییر شیب آن (θ_1). اگر $\theta_1 = 0$ را اجباری کنیم، مدل تنها یک درجه آزادی خواهد داشت و صرفاً می‌تواند خط را به بالا یا پایین حرکت دهد تا به داده‌ها نزدیک شود. نتیجه، مدلی بسیار ساده و تقریباً برابر با میانگین داده‌ها خواهد بود. در مقابل، اگر به الگوریتم اجازه تغییر θ_1 را بدهیم ولی آن را کوچک نگه

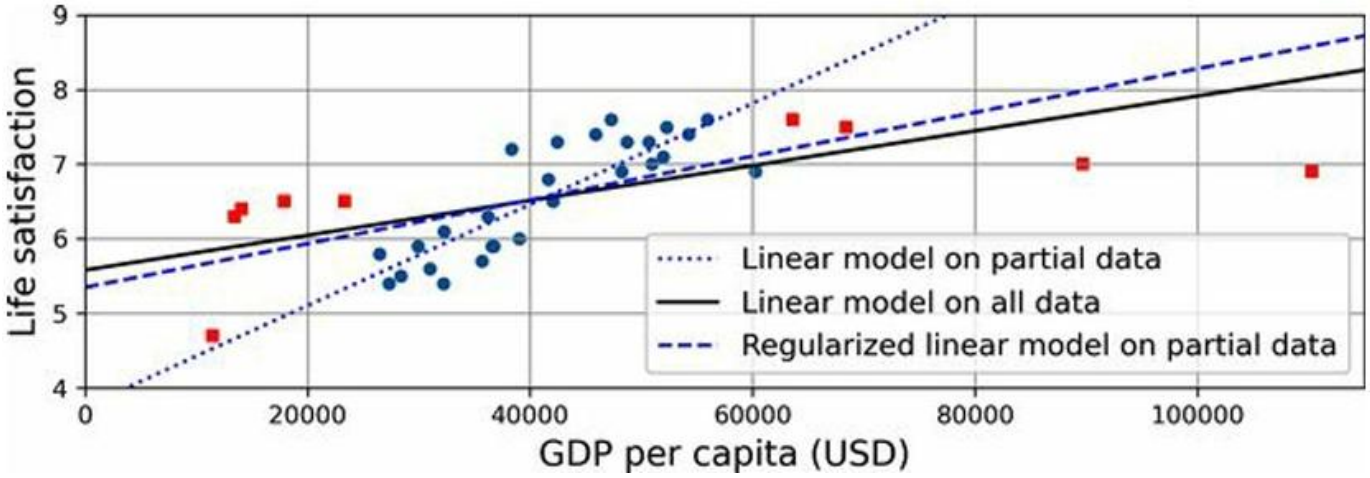
© 2025 Mohsen Zare. All rights reserved.

داریم، مدل حالتی بین یک و دو درجه آزادی خواهد داشت: ساده‌تر از مدل کامل دو درجه آزادی، اما پیچیده‌تر از مدل با یک درجه آزادی. هدف، یافتن «تعادلی بهینه» بین برازش دقیق داده‌های آموزشی و حفظ سادگی مدل به اندازه‌ای است که تعمیم‌پذیری آن به داده‌های جدید حفظ شود.

شکل ۲۴-۱ سه مدل متفاوت را مقایسه می‌کند:

- خط نقطه‌چین :مدل اولیه که فقط با داده‌های نمایش‌داده‌شده به صورت دایره آموزش دیده است (کشورهای مربعی‌شکل در آموزش استفاده نشده‌اند).
- خط ممتد :مدل دوم که با تمام داده‌ها، شامل دایره‌ها و مربع‌ها، آموزش دیده است.
- خط‌چین :مدلی که با همان داده‌های مدل اول آموزش داده شده، اما تحت محدودیت منظم‌سازی قرار گرفته است.

همان‌طور که دیده می‌شود، منظم‌سازی باعث کاهش شیب مدل شده است. نتیجه این است که اگرچه مدل جدید به اندازه مدل اول روی داده‌های آموزشی (دایره‌ها) برازش کامل ندارد، اما در پیش‌بینی داده‌های جدید و داده‌های ندیده(مربع‌ها) عملکرد بهتری دارد و تعمیم‌پذیری بالاتری ارائه می‌کند.



شکل ۲۴-۱ .منظم‌سازی با ساده‌تر کردن مدل، ریسکِ بیش‌برازش را کاهش می‌دهد و باعث بهبود تعمیم‌پذیری آن روی داده‌های جدید می‌شود.

میزان منظم‌سازی با یک «هایپرپارامتر (*hyperparameter*)»—اَبَرمؤلفه یا اَبَرمقدار؛ که برخلاف پارامترهای مدل (که از داده آموخته می‌شوند) توسط پژوهشگر یا الگوریتم تنظیم انتخاب می‌شود و نقش بسیار کلیدی در کیفیت تعمیم‌پذیری و عملکرد مدل دارد—کنترل می‌شود؛ مقادیر قابل تنظیمی

© 2025 Mohsen Zare. All rights reserved.

که قبل‌از آموزش مدل یادگیری ماشین تعیین می‌شوند، بر عملکرد مدل تأثیر می‌گذارند و در طول فرایند ثابت می‌ماند. این هایپرپارامتر بخشی از الگوریتم یادگیری است، نه خود مدل، و تعیین می‌کند تا چه اندازه مدل باید ساده نگه داشته شود. اگر مقدار آن خیلی بزرگ باشد، مدل تقریباً مسطح شده و خطر بیش‌برازش به صفر نزدیک می‌شود، اما احتمال یافتن یک راه‌حل مناسب هم کاهش می‌یابد. به همین دلیل، «تنظیم دقیق هایپرپارامترها» یکی از مراحل کلیدی در ساخت یک سیستم یادگیری ماشین موفق است. در فصل بعدی به طور مفصل مثال هایی در این مورد بررسی خواهند شد.

© 2025 Mohsen Zare. All rights reserved.

کم‌برازشِ داده‌های آموزشی

«کم‌برازش ($Underfitting$)»—تطبیق کم‌تر از حد یا برازش ناکافی با داده‌های آموزشی—نقطه‌ی مقابل«بیش‌برازش ($Overfitting$)» است. این حالت زمانی رخ می‌دهد که مدل شما برای یادگیری ساختار اصلی داده‌ها و شناسایی الگوها بیش از حد ساده باشد. به عنوان مثال، یک مدل خطی برای پیش‌بینی «رضایت از زندگی» ممکن است بسیار ساده باشد و نتواند پیچیدگی واقعی داده‌ها را پوشش دهد. در نتیجه، پیش‌بینی‌های آن حتی روی داده‌های آموزشی دقیق نخواهند بود.

برای رفع کم‌برازش، می‌توان اقدامات زیر را انجام داد:

- انتخاب یک مدل قدرتمندتر با تعداد پارامترهای بیشتر
- ارائه ویژگی‌های مرتبط‌تر و بهتر به الگوریتم یادگیری ($Feature\ Engineering$).
- کاهش محدودیت‌های مدل، مانند کاهش مقدار «هایپرپارامتر منظم‌سازی».

© 2025 Mohsen Zare. All rights reserved.

قدمی به عقب

تا اینجا مطالب زیادی درباره یادگیری ماشین پوشش داده‌ایم و ممکن است کمی احساس سردرگمی کنید. بیایید یک قدم عقب برویم و تصویر کلی را مرور کنیم:

- یادگیری ماشین یعنی اینکه ماشین‌ها بتوانند با یادگیری از داده‌ها، بدون نیاز به کدگذاری صریحِ قوانین، در انجام برخی کارها بهتر شوند.
- سیستم‌های یادگیری ماشین انواع مختلفی دارند: تحت نظارت یا بدون نظارت، دسته‌ای یا آنلاین، مبتنی بر نمونه یا مبتنی بر مدل.
- در یک پروژه یادگیری ماشین، شما داده‌ها را جمع‌آوری می‌کنید و آن‌ها را به یک الگوریتم یادگیری می‌دهید. اگر الگوریتم مبتنی بر مدل باشد، پارامترهایی را تنظیم می‌کند تا مدل با داده‌ها برازش پیدا کند(الگوهای داده را شناسایی کند) و پیش‌بینی‌های خوبی روی نمونه‌های جدید ارائه دهد. اگر الگوریتم مبتنی بر نمونه باشد، فقط مثال‌ها را حفظ می‌کند و با استفاده از معیار شباهت، آن‌ها را به نمونه‌های جدید تعمیم می‌دهد.
- کیفیت داده‌ها بسیار مهم است: اگر مجموعه آموزشی کوچک، غیرنماینده، نویزی یا پر از ویژگی‌های نامرتبط باشد(garbage in, garbage out)، سیستم عملکرد خوبی نخواهد داشت.
- مدل شما نباید خیلی ساده باشد (که باعث Underfitting می‌شود) و نه خیلی پیچیده (که باعث Overfitting می‌شود).

یک نکته مهم دیگر باقی مانده است: پس از آموزش مدل، شما نمی‌خواهید فقط امیدوار باشید که مدل بتواند روی داده‌های جدید هم پیش‌بینی خوبی انجام دهد. شما باید عملکرد آن را ارزیابی کنید و در صورت نیاز، آن را «تنظیم دقیق (*fine-tune*)» کنید. بیایید مراحل انجام این کار را بررسی کنیم.

© 2025 Mohsen Zare. All rights reserved.

آزمایش و اعتبارسنجی

تنها راه برای فهمیدن اینکه یک مدل چقدر خوب می‌تواند روی داده‌های جدید پیش‌بینی کند، «آزمودن آن روی داده‌های جدید» است. یک روش ساده، قرار دادن مدل در مرحله تولید و زیر نظر گرفتن عملکرد آن است. این روش ممکن است کار کند، اما اگر مدل به‌طرز قابل توجهی ضعیف باشد، کاربران از نتایج ناراضی خواهند شد؛ بنابراین این روش همیشه ایده‌آل نیست.

روش بهتر این است که داده‌ها را به دو مجموعه تقسیم کنید: «مجموعه‌ی آموزشی (*training set*)» و «مجموعه‌ی آزمون (*test set*)». همانطور که از نام‌ها مشخص است، شما مدل را با استفاده از مجموعه آموزشی آموزش می‌دهید و سپس با مجموعه آزمون عملکرد آن را آزمایش(تست) و ارزیابی می‌کنید. میزان خطایی که مدل روی داده‌هایی که در آموزش استفاده نشده‌اند (مثل داده آزمون یا داده واقعی جدید) دارد، «خطای تعمیم (*generalization error*)» یا «خطای خارج از نمونه (*out-of-sample error*)» نامیده می‌شود. با بررسی مدل روی مجموعه آزمون، می‌توان تخمینی از این خطا به دست آورد که نشان می‌دهد مدل در مواجهه با داده‌های دیده‌نشده چقدر خوب عمل می‌کند.

اگر مدل روی مجموعه آموزشی خطای کمی داشته باشد اما خطای تعمیم زیاد باشد، این نشانه‌ی بیش‌برازش مدل روی داده‌های آموزشی است.

نکته

معمولاً حدود ۸۰٪ از داده‌ها برای آموزش (training) و ۲۰٪ برای آزمایش (test) در نظر گرفته یا نگه‌داشته (*hold out*) می‌شود. با این حال، نسبتِ دقیق به اندازه‌ی مجموعه‌داده‌ها بستگی دارد. برای مثال، اگر مجموعه داده شامل ۱۰ میلیون نمونه باشد، حتی در نظر گرفتن ۱٪ از داده‌ها برای مجموعه آزمون یعنی ۱۰۰,۰۰۰ نمونه خواهید داشت، که معمولاً برای به دست آوردن تخمین دقیق خطای تعمیم کافی است.

© 2025 Mohsen Zare. All rights reserved.

تنظیم هایپرپارامتر و انتخاب مدل

ارزیابی یک مدل به ظاهر ساده است: کافی است آن را روی مجموعه آزمون امتحان کنید. اما فرض کنید بین دو نوع مدل، مثلاً یک «مدل خطی (*linear model*)» و یک «مدل چندجمله‌ای (*polynomial model*)» مُردد هستید. چگونه تصمیم بگیرید؟ یک روش این است که هر دو مدل را آموزش داده و با استفاده از مجموعه‌ی آزمون «میزان تعمیم‌پذیری (*generalization*)» آنها را مقایسه کنید.

حال فرض کنید مدل خطی بهتر تعمیم می‌یابد، اما شما می‌خواهید مقداری منظم‌سازی اعمال کنید تا از بیش‌برازش جلوگیری شود. سوال این است: چگونه مقدار «هایپرپارامتر منظم‌سازی (*regularization hyperparameter*)» را انتخاب کنیم؟ یکی از روش‌ها این است که ۱۰۰ مدل مختلف با ۱۰۰ مقدار متفاوت هایپرپارامتر آموزش دهید. فرض کنید بهترین مقدار را پیدا می‌کنید که مدل با کمترین خطای تعمیم، مثلاً ۵% خطا، ایجاد می‌کند. سپس این مدل را وارد فاز تولید می‌کنید، اما متأسفانه عملکرد آن مطابق انتظار نیست و خطای آن ۱۵% است. حالا سؤال این است: چه اتفاقی افتاده است؟

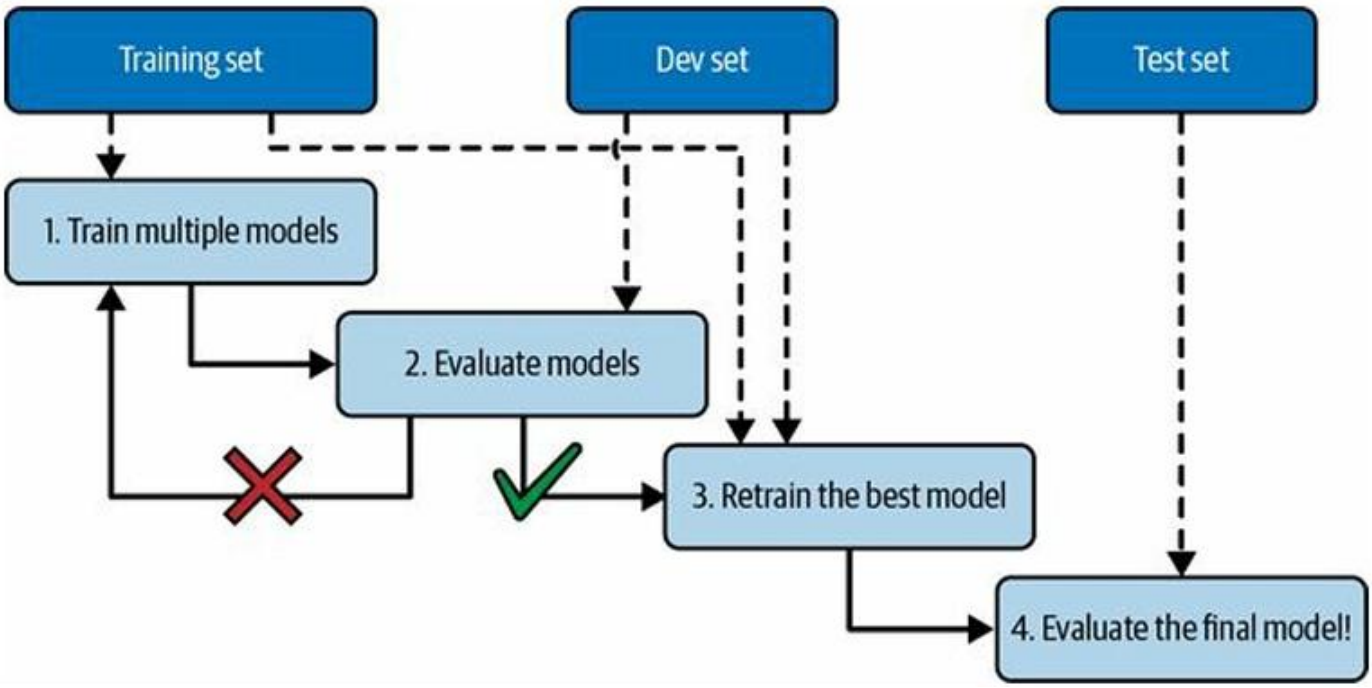
مشکل اینجاست که شما خطای تعمیم را چندین بار روی همان مجموعه‌ی آزمون اندازه‌گیری کرده‌اید و مدل و هایپرپارامترها را به گونه‌ای انتخاب کرده‌اید که بهترین عملکرد را روی آن مجموعه خاص داشته باشند. به عبارت دیگر، مدل شما «برای همان مجموعه‌ی آزمون» بهینه شده است و این باعث می‌شود که احتمالاً روی داده‌های جدید نتایج مشابهی نداشته باشد.

یک راه حل رایج برای این مشکل(خطای تعمیم و کم‌برازش)، استفاده از «اعتبارسنجی با داده‌ی نگه‌داشته‌شده (*holdout validation*)» است (شکل ۱-۲۵). در این روش، بخشی از مجموعه آموزشی را برای ارزیابی مدل‌های کاندید و انتخاب بهترین مدل نگه می‌دارید. این بخش نگه‌داشته‌شده، «مجموعه اعتبارسنجی (*validation set*)» یا «مجموعه توسعه (*development set* یا *dev set*)» نامیده می‌شود.

© 2025 Mohsen Zare. All rights reserved.

به طور مشخص، چندین مدل با مقادیر مختلف هایپرپارامتر روی «مجموعه آموزشیِ کاهش‌یافته» (یعنی مجموعه آموزشی اصلی منهای مجموعه‌ی اعتبارسنجی) آموزش داده می‌شوند. سپس مدلی انتخاب می‌شود که در مجموعه اعتبارسنجی بهترین عملکرد را داشته باشد.

پس از اتمام این مرحله—یعنی فرآیند «اعتبارسنجی با داده‌ی نگه‌داشته‌شده»—، بهترین مدل دوباره روی کل مجموعه آموزشی (شامل مجموعه اعتبارسنجی) آموزش داده می‌شود و مدل نهایی به دست می‌آید. در نهایت، مدل نهایی روی مجموعه آزمون ارزیابی می‌شود تا تخمینی دقیق از خطای تعمیم به دست آید.



شکل ۱-۲۵. انتخاب مدل با استفاده از اعتبارسنجی داده‌ی نگه‌داشته‌شده

این روش معمولاً عملکرد خوبی دارد. با این حال، اگر «مجموعه اعتبارسنجی (*validation set*)» بسیار کوچک باشد، ارزیابی‌های مدل دقت کافی ندارند و ممکن است به اشتباه یک مدل غیر بهینه انتخاب شود. از سوی دیگر، اگر مجموعه اعتبارسنجی خیلی بزرگ باشد، مجموعه آموزشیِ باقی‌مانده برای آموزش مدل نهایی بسیار کوچک خواهد شد. این یک اتفاقِ مشکل‌ساز است، زیرا مدل نهایی روی کل داده‌ها آموزش داده می‌شود و مقایسه مدل‌های کاندید که با داده‌های محدود آموزش دیده‌اند، نتایج قابل اعتمادی نمی‌دهد. یک راه حل برای این مشکل، انجام «اعتبارسنجی متقاطع

© 2025 Mohsen Zare. All rights reserved.

($cross\text{-}validation$)» مکرر با استفاده از «چند مجموعه اعتبارسنجی کوچک» است. در این روش، هر مدل پس از آموزش روی باقی داده‌ها، یک بار در هر مجموعه اعتبارسنجی ارزیابی می‌شود. با میانگین گرفتن از تمام این ارزیابی‌ها، تصویر دقیق‌تری از عملکرد واقعی مدل به دست می‌آید. با این حال، یک نکته مهم وجود دارد: هر بار ارزیابی اضافه، زمان آموزش را افزایش می‌دهد و فرآیند طولانی‌تر خواهد شد.

© 2025 Mohsen Zare. All rights reserved.

عدم تطابق داده‌ها

گاهی حجم زیادی از داده‌ها برای آموزش مدل به‌راحتی در دسترس است، اما این داده‌ها احتمالاً به‌طور کامل نماینده(مُعرف) داده‌هایی که در «تولید(production)» استفاده می‌شوند، نیستند. برای نمونه، فرض کنید می‌خواهید یک برنامه تلفن همراه بسازید که از گل‌ها عکس بگیرد و به‌طور خودکار گونه‌ی آنها را شناسایی کند. ممکن است بتوانید میلیون‌ها عکس گل را از فضای وب دانلود کنید، اما این تصاویر به‌طور دقیق نماینده‌ی تصاویری نیستند که در واقع با اپلیکیشن روی دستگاه تلفن همراه گرفته خواهند شد. در مقابل، شاید فقط ۱,۰۰۰ عکس «نماینده‌ی» واقعی در اختیار داشته باشید که با خود اپلیکیشن ثبت شده‌اند.

در این زمینه، مهم‌ترین قانونی که باید به خاطر داشته باشید این است که هم «مجموعه اعتبارسنجی» و هم «مجموعه آزمون» باید تا حد امکان نماینده‌ی داده‌هایی باشند که انتظار دارید در تولید استفاده شوند. بنابراین، این مجموعه‌ها باید منحصراً از تصاویر «نماینده» تشکیل شده باشند. می‌توانید تصاویر را «دَرهَم(shuffle)» کنید و نیمی را در مجموعه اعتبارسنجی و نیمی را در مجموعه آزمون قرار دهید، با این اطمینان که هیچ مورد تکراری یا تقریباً تکراری در هردو مجموعه وجود نداشته باشد. پس از آموزش مدل روی تصاویر وب، اگر مشاهده کنید که عملکرد مدل در مجموعه اعتبارسنجی نااُمیدکننده است، نمی‌توان تشخیص داد که آیا این مشکل ناشی از بیش‌برازش مدل روی مجموعه آموزش است یا صرفاً به دلیل عدم تطابق بین تصاویر وب و تصاویر گرفته‌شده با اپلیکیشن تلفن همراه است.

یک راه‌حل این است که بخشی از تصاویر آموزشی وب را در مجموعه دیگری نگه دارید که اندرو اِنگ(Andrew Ng) آن را «مجموعهٔ آموزش-توسعه (*train-dev set*)» نامیده است (شکل ۱-۲۶). پس از آموزش مدل (روی مجموعه آموزشی، «نه» روی مجموعه‌ی آموزش-توسعه) می‌توانید عملکرد آن را در مجموعه‌ی آموزش-توسعه ارزیابی کنید. اگر مدل عملکرد ضعیفی داشته باشد، یعنی بیش‌برازش در مجموعه آموزش-توسعه رخ داده است و لازم است مدل را ساده یا منظم‌تر کنید، داده‌های آموزشی بیشتری جمع‌آوری کنید و داده‌های موجود را پاک‌سازی نمایید. اما اگر مدل در مجموعه آموزش-توسعه عملکرد خوبی داشته باشد، می‌توانید آن را در «مجموعه‌ی توسعه» ارزیابی کنید. اگر در این مرحله عملکرد ضعیف بود، مشکل ناشی از عدم تطابق یا برازش داده‌ها است. برای رفع این مشکل، می‌توان تصاویر وب را پیش‌پردازش کرد تا بیشتر شبیه تصاویری شوند که با اپلیکیشن تلفن همراه ثبت می‌شوند و سپس مدل را مجدداً آموزش داد. هنگامی که مدلی داشته باشید که هم در مجموعه آموزش-توسعه

© 2025 Mohsen Zare. All rights reserved.

و هم در مجموعه توسعه عملکرد خوبی دارد، می‌توانید آن را برای آخرین بار در «مجموعه آزمون» ارزیابی کنید تا میزان کارایی احتمالی آن در محیط عملیاتی مشخص شود.

شکل ۱-۲۶. هنگامی که داده‌های واقعی محدود هستند (تصویر سمت راست)، می‌توان از داده‌های مشابه و فراوان (تصویر سمت چپ) برای آموزش مدل استفاده کرد و بخشی از آنها را در «مجموعه آموزش-توسعه» نگه داشت تا میزان بیش‌برازش ارزیابی شود. سپس، داده‌های واقعی برای ارزیابی عدم تطابق داده‌ها در «مجموعه توسعه» و همچنین ارزیابی عملکرد نهایی مدل در «مجموعه آزمون» به کار می‌روند.

© 2025 Mohsen Zare. All rights reserved.

قضیه‌ی «هیچ ناهار رایگانی وجود ندارد»

یک مدل، نسخه ساده‌شده‌ای از داده‌ها است که با هدف حذف جزئیات غیرضروری و کم‌اهمیت ایجاد می‌شود؛ جزئیاتی که بعید است در تعمیم به نمونه‌های جدید مؤثر باشند. انتخاب یک نوع مشخص از مدل، به معنای پذیرش ضمنی یک سری «فرضیات($assumptions$)» درباره ماهیت داده‌ها است. برای نمونه، با انتخاب یک مدل خطی، فرض می‌کنید که داده‌ها در اصل ساختاری خطی دارند و اختلاف بین نقاط داده و خط مستقیم صرفاً نویز است که می‌توان آن را نادیده گرفت.

«دیوید ولپرت» در مقاله‌ای مشهور در سال ۱۹۹۶ نشان داد که اگر هیچ فرضی درباره داده‌ها نداشته باشید، هیچ دلیلی وجود ندارد که یک مدل را بر مدل دیگر ترجیح دهید. این نتیجه با عنوان «هیچ ناهار رایگانی وجود ندارد (NFL یا $No\ Free\ Lunch$)» شناخته می‌شود. برای برخی مجموعه‌داده‌ها، بهترین انتخاب یک مدل خطی است، در حالی که برای برخی دیگر، یک شبکه عصبی عملکرد بهتری دارد. هیچ مدلی وجود ندارد که «از پیش تضمین شود» همیشه بهتر عمل کند (نام این قضیه نیز به همین دلیل انتخاب شده است). تنها روش قطعی برای فهمیدن بهترین مدل، ارزیابی همه گزینه‌ها است، اما از آن‌جا که این کار در عمل ممکن نیست، معمولاً با تکیه بر فرضیات معقول درباره داده‌ها، تنها چند مدل مناسب را ارزیابی می‌کنید. برای مثال، در مسائل ساده می‌توان مدل‌های خطی را با درجات مختلف منظم‌سازی بررسی کرد، و در مسائل پیچیده‌تر، انواع مختلف شبکه‌های عصبی را مورد ارزیابی قرار داد.

© 2025 Mohsen Zare. All rights reserved.

تمرین‌ها

در این فصل، برخی از مهم‌ترین مفاهیم «یادگیری ماشین» مرور شد. در فصل‌های بعد، این مباحث را عمیق‌تر بررسی کرده و کدهای بیشتری خواهیم نوشت. پیش از آن، اطمینان حاصل کنید که می‌توانید به پرسش‌های زیر پاسخ دهید:

۱. یادگیری ماشین را چگونه تعریف می‌کنید؟

۲. چهار نمونه از کاربردهایی که در آن یادگیری ماشین عملکرد برجسته‌ای دارد را نام ببرید.

۳. «مجموعه آموزشی برچسب‌گذاری‌شده» چیست؟

٤. دو مسئله‌ی رایج در «یادگیری نظارت‌شده» کدام‌اند؟ آیا می‌توانید چهار مورد از مسائل رایج بدون نظارت را نام ببرید؟

۵. برای آموزش یک ربات به راه رفتن در زمین‌های ناشناخته، از چه نوع الگوریتمی استفاده می‌شود؟

۶. برای تقسیم مشتریان به چند گروه مجزا، کدام نوع الگوریتم به‌کار می‌رود؟

۷. آیا تشخیص هرزنامه یک مسئله نظارت‌شده است یا نظارت‌نشده؟

۸. «سیستم یادگیری آنلاین» چیست؟

۹. «یادگیری خارج از هسته» به چه معناست؟

۱۰. کدام نوع الگوریتم برای پیش‌بینی به معیار شباهت متکی است؟

۱۱. تفاوت میان «پارامتر مدل» و «هایپرپارامتر مدل» چیست؟

۱۲. الگوریتم‌های مبتنی بر مدل به دنبال چه هستند؟ رایج‌ترین استراتژی آن‌ها برای دستیابی به موفقیت چیست؟ و این الگوریتم‌ها چگونه پیش‌بینی انجام می‌دهند؟

۱۳. چهار مورد از چالش‌های اصلی در یادگیری ماشین را بیان کنید.

© 2025 Mohsen Zare. All rights reserved.

۱۴. اگر مدلی روی داده‌های آموزشی عملکرد مطلوبی داشته باشد اما در تعمیم به نمونه‌های جدید ضعیف عمل کند، چه وضعیتی رخ داده است؟ سه راه‌حل ممکن برای این مشکل را نام ببرید.

۱۵. «مجموعه آزمون» چیست و چرا از آن استفاده می‌شود؟

۱۶. هدف از «مجموعه اعتبارسنجی» چیست؟

۱۷. «مجموعه آموزش-توسعه» چیست، چه زمانی به آن نیاز است و چگونه مورد استفاده قرار می‌گیرد؟

۱۸. اگر هایپرپارامترها با استفاده از مجموعه آزمون تنظیم شوند، چه مشکلی ممکن است به‌وجود آید؟

پاسخ این تمرین‌ها در انتهای نوت‌بوک این فصل، در آدرس *https://homl.info/colab3* موجود است.

© 2025 Mohsen Zare. All rights reserved.

فصل ۲. پروژه یادگیری ماشین از ابتدا تا انتها

در این فصل، یک پروژه نمونه را از ابتدا تا انتها بررسی خواهید کرد و نقش یک دانشمند داده تازه‌استخدام شده در یک شرکت املاک و مستغلات را ایفا می‌کنید. این مثال ساختگی است و هدف آن، نشان دادن مراحل اصلی یک پروژه یادگیری ماشین است، نه یادگیری جزئیات مربوط به تجارت املاک و مستغلات.

مراحل اصلی که در این فصل بررسی می‌شوند عبارتند از:

۱. بررسی چشم‌انداز و دورنما.
۲. دریافت داده‌ها
۳. کاوش و مصورسازی داده‌ها برای استخراج بینش.
۴. آماده‌سازی داده‌ها برای الگوریتم‌های یادگیری ماشین.
۵. انتخاب مدل و آموزش آن.

© 2025 Mohsen Zare. All rights reserved.

۶. تنظیم دقیق مدل.

۷. ارائه راه‌حل.

۸. راه‌اندازی، نظارت و نگهداری سیستم.

© 2025 Mohsen Zare. All rights reserved.

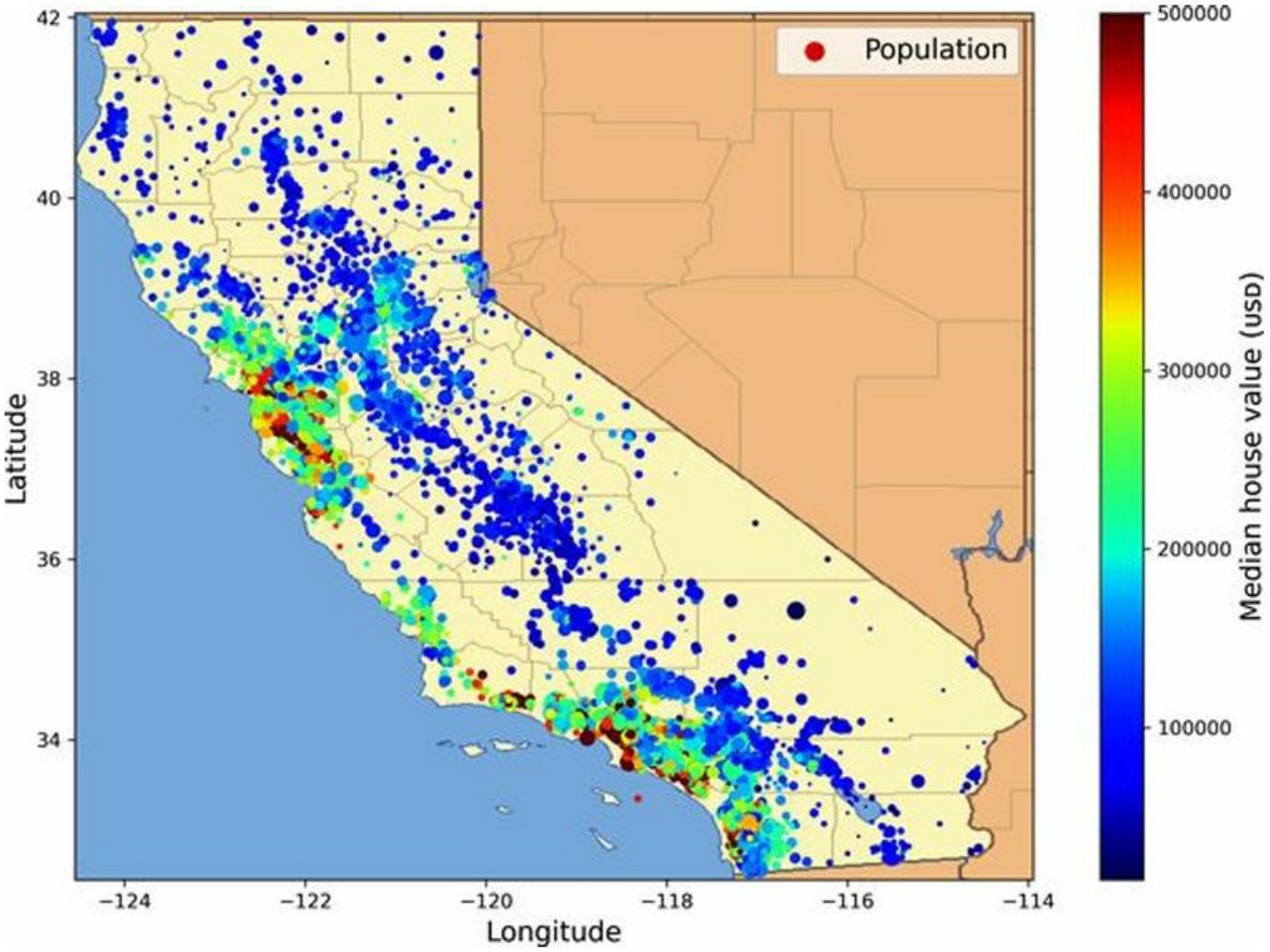
کار با داده‌های واقعی

هنگام یادگیری «یادگیری ماشین»، بهتر است با داده‌های دنیای واقعی تمرین کنید تا با مجموعه داده‌های مصنوعی. خوشبختانه هزاران مجموعه داده باز در دسترس هستند که در حوزه‌های مختلف قرار دارند. در ادامه چند منبع برای دسترسی به این داده‌ها آمده است:

- ریپازیتوری‌های داده‌ی باز محبوب (open data Repository):
  - OpenML.org
  - Kaggle.com
  - PapersWithCode.com
  - مخزن یادگیری ماشین UC Irvine
  - مجموعه داده‌های AWS آمازون
  - مجموعه داده‌های TensorFlow
- متا پورتال‌ها (که ریپازیتوری‌های داده باز را فهرست می‌کنند):
  - DataPortals.org
  - OpenDataMonitor.eu
- سایر صفحات مرجع که فهرستی از ریپو های محبوب ارائه می‌دهند:
  - لیست مجموعه داده‌های یادگیری ماشین در ویکی‌پدیا
  - Quora.com
  - مجموعه داده‌ها در subreddit

© 2025 Mohsen Zare. All rights reserved.

در این فصل از «مجموعه داده‌های قیمت مسکن کالیفرنیا» از ریپازیتوری StatLib استفاده خواهیم کرد (شکل ۲-۱ را ببینید). این داده‌ها بر اساس سرشماری کالیفرنیا در سال ۱۹۹۰ جمع‌آوری شده‌اند. هرچند این مجموعه داده جدید نیست (یک خانه خوب در منطقه خلیج در آن زمان هنوز مقرون‌به‌صرفه بود)، اما دارای ویژگی‌های متنوعی برای یادگیری است، بنابراین فرض می‌کنیم که این داده‌ها تازه هستند. برای اهداف آموزشی، یک ویژگی دسته‌بندی اضافه شده و برخی ویژگی‌ها حذف شده‌اند.



شکل ۲-۱. قیمت مسکن در کالیفرنیا

© 2025 Mohsen Zare. All rights reserved.

به چشم انداز و دور نما نگاه کنید

به «شرکت یادگیری ماشینی مسکن» خوش‌آمدید! مأموریت اولیه شما استفاده از داده‌های سرشماری کالیفرنیا برای ساخت مدلی از قیمت مسکن در ایالت است. این داده‌ها شامل معیارهایی مانند جمعیت، درآمد متوسط و قیمت متوسط مسکن برای هر «گروه بلوکی» در کالیفرنیا هستند. گروه‌های بلوکی کوچک‌ترین واحد جغرافیایی هستند که اداره سرشماری ایالات متحده داده‌های نمونه را برای آن‌ها منتشر می‌کند (هر گروه بلوکی معمولاً جمعیتی بین ۶۰۰ تا ۳,۰۰۰ نفر دارد). برای سهولت، این گروه‌ها را به اختصار «منطقه» می‌نامیم.

مدل شما باید از این داده‌ها بیاموزد و قادر باشد با توجه به سایر معیارها، «میانگین قیمت مسکن» در هر منطقه را پیش‌بینی کند.

نکته

از آنجا که شما یک دانشمند داده‌ی منظم و دقیق هستید، اولین قدم، آماده کردن «چک‌لیست پروژه یادگیری ماشین» است. می‌توانید کار را با چک‌لیست پیوست الف آغاز کنید؛ این چک‌لیست برای اغلب پروژه‌های یادگیری ماشین مناسب است، اما باید آن را متناسب با نیازهای خاص خود تنظیم کنید. در این فصل، بسیاری از موارد این چک‌لیست بررسی خواهند شد، در حالی که برخی از آن‌ها نادیده گرفته می‌شوند—یا به دلیل عدم نیاز به توضیح، یا اینکه در فصل‌های بعدی به آنها پرداخته خواهد شد.

© 2025 Mohsen Zare. All rights reserved.

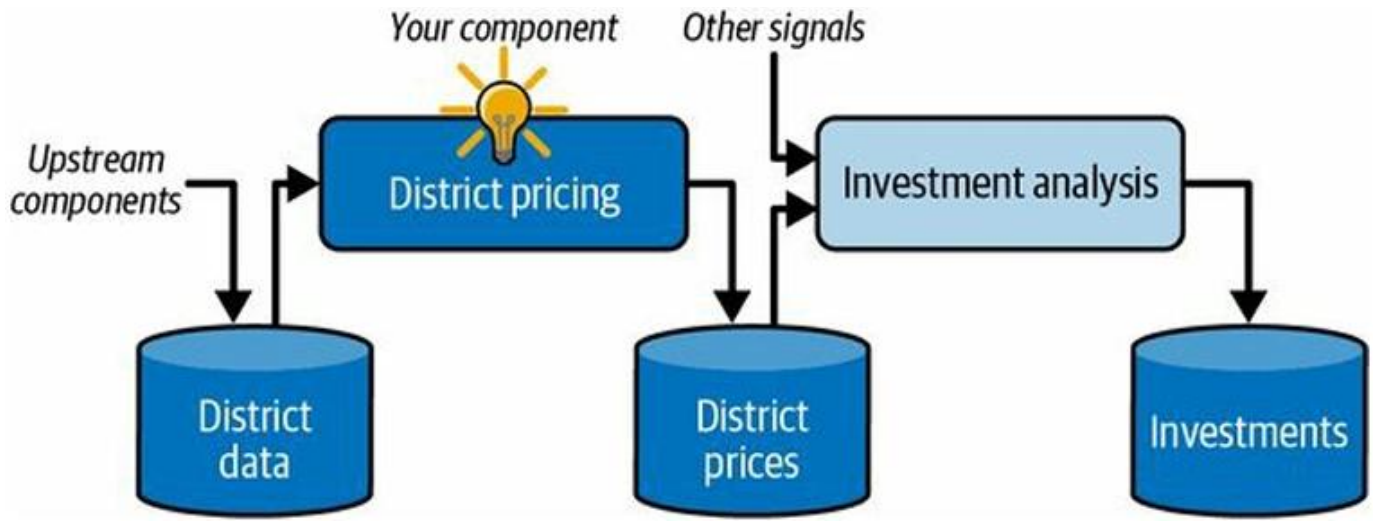
چارچوب‌بندی مسئله

اولین پرسشی که باید از مدیر خود بپرسید این است که «هدف کسب‌وکار دقیقاً چیست». ساخت یک مدل به‌تنهایی معمولاً هدف نهایی نیست؛ بلکه شرکت می‌خواهد بداند چطور از مدل استفاده کند و از آن بهره‌مند شود. درک دقیق هدف اهمیت زیادی دارد، زیرا مشخص می‌کند چگونه مسئله را چارچوب‌بندی خواهید کرد، کدام الگوریتم‌ها مناسب هستند، از کدام معیار عملکرد برای ارزیابی مدل استفاده خواهید کرد و چه میزان تلاش برای بهبود آن صرف خواهید شد.

مدیرتان پاسخ می‌دهد که خروجی مدل شما—یعنی پیش‌بینی «میانگین قیمت مسکن» در یک منطقه—همراه با سایر سیگنال‌ها به یک سیستم یادگیری ماشین دیگر داده خواهد شد (شکل ۲-۲ را ببینید). این سیستمِ پایین‌دستی تصمیم می‌گیرد که آیا سرمایه‌گذاری در یک منطقه خاص ارزش دارد یا خیر. انجام درست این فرآیند اهمیت بالایی دارد، زیرا مستقیماً بر درآمد شرکت تأثیر می‌گذارد.

سؤال بعدی که باید از مدیر خود بپرسید این است که «راه‌حل فعلی، در صورت وجود، چگونه اجرا می‌شود». وضعیت موجود معمولاً مرجعی برای سنجش عملکرد فراهم می‌کند و بینش‌هایی درباره نحوه حل مسئله ارائه می‌دهد. مدیر شما پاسخ می‌دهد که در حال حاضر «قیمت مسکن در هر منطقه» توسط کارشناسان به‌صورت دستی تخمین زده می‌شود: یک تیم اطلاعاتِ به‌روز منطقه را جمع‌آوری می‌کند و در مواردی که امکان به‌دست آوردن قیمت متوسط مسکن وجود ندارد، آن را با استفاده از قوانین پیچیده تخمین می‌زنند.

© 2025 Mohsen Zare. All rights reserved.



شکل ۲-۲. یک مسیر یادگیری ماشین برای سرمایه‌گذاری در املاک و مستغلات

این روش پرهزینه و زمان‌بر است و تخمین‌های آن‌ها چندان دقیق نیست؛ در مواردی که موفق به تعیین قیمت واقعی مسکن می‌شوند، اغلب متوجه می‌شوند که تخمین‌هایشان بیش از ۳۰ درصد خطا داشته است. به همین دلیل، شرکت معتقد است که آموزش مدلی برای پیش‌بینی «قیمت متوسط مسکن» در هر منطقه، با توجه به سایر داده‌های مرتبط، مفید خواهد بود. داده‌های سرشماری به‌نظر می‌رسد مجموعه داده‌ای بسیار مناسب برای این هدف باشد، زیرا شامل قیمت متوسط مسکن هزاران منطقه و سایر اطلاعات مرتبط است.

© 2025 Mohsen Zare. All rights reserved.

pipline داده

دنباله‌ای از اجزای پردازش داده، «خطِ سِیر پردازش داده» یا «خط لوله پردازش داده (*pipeline*)» نامیده می‌شود. این pipline ها در سیستم‌های یادگیری ماشین بسیار رایج هستند، زیرا حجم داده‌ها زیاد است و تبدیل‌های متعددی باید روی آن‌ها اعمال شود.

این اجزاء معمولاً به‌صورت ناهمزمان اجرا می‌شوند. هر جزء مقدار زیادی داده دریافت می‌کند، آن را پردازش می‌کند و خروجی را در یک مخزن داده منتشر می‌نماید. سپس، مدتی بعد، جزء بعدی در خط لوله پردازش این داده‌ها را دریافت کرده و خروجی خود را تولید می‌کند. هر جزء نسبتاً مستقل است و رابط بین اجزاء صرفاً همان مخزن داده است. این ساختار درک سیستم را ساده می‌کند (با استفاده از گراف جریان داده) و امکان تمرکز تیم‌های مختلف روی اجزای متفاوت را فراهم می‌آورد. علاوه بر این، اگر یک جزء خراب شود، اجزاءِ پایین‌دستی معمولاً می‌توانند (حداقل برای مدتی) تنها با استفاده از آخرین خروجی جزء خراب، به عملکرد عادی خود ادامه دهند. این ویژگی، معماری سیستم را بسیار مقاوم(robust) می‌سازد.

با این حال، در صورت فقدان نظارت مناسب، یک جزء خراب ممکن است مدتی نادیده گرفته شود. در این صورت، داده‌ها قدیمی شده و عملکرد کلی سیستم اُفت می‌کند.

با تمام این اطلاعات، اکنون آماده‌اید طراحی سیستم تان را آغاز کنید. ابتدا مشخص کنید مدل به چه نوع «نظارت آموزشی» نیاز دارد: آیا این یک مسئله «یادگیری نظارت‌شده»، «یادگیری بدون نظارت»، «نیمه‌نظارتی»، «خودنظارتی» یا «یادگیری تقویتی» است؟ همچنین تعیین کنید که مسئله از نوع «طبقه‌بندی»، «رگرسیون» یا نوع دیگری است. آیا باید از تکنیک‌های «یادگیری دسته‌ای» استفاده کنید یا «یادگیری آنلاین» مناسب‌تر است؟ پیش از ادامه مطالعه، مکث کنید و سعی کنید خودتان به این سؤالات پاسخ دهید.

© 2025 Mohsen Zare. All rights reserved.

آیا پاسخ‌ها را پیدا کرده‌اید؟ بیایید بررسی کنیم. این مثال به‌وضوح یک مسئله «یادگیری نظارت‌شده» است، زیرا مدل می‌تواند با استفاده از «نمونه‌های برچسب‌گذاری‌شده (*labeled examples*)» آموزش داده شود؛ هر نمونه دارای خروجی مورد انتظار، یعنی «میانگین قیمت مسکن» در منطقه است. این یک مسئله رگرسیون معمولی است، زیرا مدل موظف است یک مقدار عددی را پیش‌بینی کند. به‌طور مشخص، این یک «رگرسیون چندگانه (*multiple regression*)» است، زیرا سیستم از چندین ویژگی برای پیش‌بینی استفاده می‌کند، مانند جمعیت منطقه و میانگین درآمد. در عین حال، این یک «رگرسیون تک‌متغیره (*univariate regression*)» است، زیرا هدف پیش‌بینی یک مقدار واحد برای هر منطقه است. اگر قصد داشتیم چندین مقدار برای هر منطقه پیش‌بینی کنیم، با یک «رگرسیون چندمتغیره (*multivariate regression*)» مواجه می‌شدیم. در نهایت، از آنجا که هیچ جریان پیوسته‌ای از داده‌ها وارد سیستم نمی‌شود، نیازی به تنظیم سریع داده‌ها نیست و حجم داده‌ها به اندازه‌ای است که در حافظه جای می‌گیرد، بنابراین استفاده از «یادگیری دسته‌ایِ» ساده به‌خوبی عمل خواهد کرد.

نکته

اگر حجم داده‌ها بسیار بزرگ بود، می‌توانستید فرآیند «یادگیری دسته‌ای (batch learning)» خود را بین چندین سرور تقسیم کنید (با استفاده از تکنیک «کاهشِ نگاشت (Map Reduce)») یا از تکنیک «یادگیری آنلاین (online learning)» بهره ببرید.

© 2025 Mohsen Zare. All rights reserved.

انتخاب یک معیار عملکرد

گام بعدی، انتخاب «معیار عملکرد (*performance measure*)» مناسب است. یکی از معیارهای رایج برای مسائل رگرسیون، « خطای جذر میانگین مربعات (*RMSE: Root Mean Square Error*)» است. این معیار ارزیابی—که بزرگی خطاها را با جذر میانگین مربعات آن‌ها می‌سنجد—میزان خطایی را که سیستم معمولاً در پیش‌بینی‌های خود ایجاد می‌کند، نشان می‌دهد و به خطاهای بزرگ وزن بیشتری می‌دهد. معادله ۲-۱، فرمول ریاضی محاسبه RMSE را نمایش می‌دهد:

معادله ۲-۱. جذر میانگین مربعات خطا (RMSE)

$$\mathrm{RMSE}(\mathrm{X}, \mathrm{h}) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\mathrm{h}(\mathrm{x}^{(\mathrm{i})}) - \mathrm{y}^{(\mathrm{i})})^2}$$

نمادگذاری‌ها

این معادله چندین «نمادگذاری (notation)» بسیار رایج در یادگیری ماشین را معرفی می‌کند که در سراسر این کتاب از آن‌ها استفاده خواهد شد:

- «m» تعداد نمونه‌ها(samples) در مجموعه داده‌ای است که RMSE روی آن‌ها محاسبه می‌شود.
  - برای مثال، اگر در حال ارزیابی RMSE روی یک «مجموعه اعتبارسنجی» شامل ۲,۰۰۰ منطقه هستید، آنگاه $m = 2000$.
- «$x^{(i)}$» برداری از تمام مقادیر ویژگی‌هایِ(features) نمونه‌ی i اُم در مجموعه داده است، به جز برچسب آن و «$y^{(i)}$» برچسب نمونه‌ی i اُم است، یعنی مقدار خروجی مورد انتظار برای آن نمونه

© 2025 Mohsen Zare. All rights reserved.

- برای مثال، اگر اولین منطقه در مجموعه داده‌ها در طول جغرافیایی °۱۱۸٫۲۹- و عرض جغرافیایی °۳۳٫۹۱ واقع شده باشد، جمعیت آن ۱,۴۱۶ نفر با درآمد متوسط ۳۸,۳۷۲ دلار و ارزش متوسط خانه ۱۵۶,۴۰۰ دلار باشد (فعلاً سایر ویژگی‌ها را نادیده می‌گیریم)، آنگاه:

$$x^{(1)} = -118.2933.911,41638,372$$

و همچنین:

$$y^{(1)} = 156,400$$

- «$\mathbf{X}$»ماتریسی است که شامل تمام مقادیر ویژگی‌ها همه نمونه‌های موجود در مجموعه داده است، به جز برچسب‌ها. برای هر نمونه یک سطر وجود دارد و سطر i اُم برابر با ترانهاده‌ی «$x^{(i)}$» است که با $\left(x^{(i)}\right)^{\mathrm{T}}$ نشان داده می‌شود.

  - برای مثال، اگر ناحیه اول همانطور که توضیح داده شد باشد، ماتریس $\mathbf{X}$ به شکل زیر خواهد بود:

$$X = \left(x^{(1)}\right)^{\mathrm{T}}\left(x^{(2)}\right)^{\mathrm{T}} \vdots \left(x^{(1999)}\right)^{\mathrm{T}}\left(x^{(2000)}\right)^{\mathrm{T}}$$
$$= -118.2933.911,41638,372 \vdots\vdots\vdots\vdots$$

- «h» تابع پیش‌بینی سیستم شما است که به آن «فرضیه ($hypothesis$)» نیز گفته می‌شود. وقتی بردار ویژگی یک نمونه «$x^{(i)}$» به سیستم داده می‌شود، مقدار پیش‌بینی‌شده «$\hat{y}^{(i)} = h\left(x^{(i)}\right)$» برای آن نمونه تولید می‌گردد («$\hat{y}$» به صورت «y-hat» تلفظ می‌شود).

  - برای مثال، اگر سیستم شما پیش‌بینی کند که «میانگین قیمت مسکن» در منطقه اول ۱۵۸,۴۰۰ دلار است، آنگاه $\hat{y}^{(1)} = h\left(x^{(1)}\right) = 158,400$. خطای پیش‌بینی برای این منطقه $\hat{y}^{(1)} - y^{(1)} = 2,000$ است.

© 2025 Mohsen Zare. All rights reserved.

- «$\mathrm{RMSE}(\mathbf{X}, h)$» تابع هزینه (cost function) است که روی مجموعه‌ی نمونه‌ها با استفاده از فرضیه h شما اندازه‌گیری می‌شود.

برای مقادیر اسکالر (مانند m یا $y^{(i)}$) و نام توابع (مانند h) از فونت کوچک ایتالیک، برای بردارها (مانند $\mathbf{x}^{(i)}$) از فونت کوچک پررنگ و برای ماتریس‌ها (مانند $\mathbf{X}$) از فونت بزرگ پررنگ استفاده می‌کنیم.

اگرچه «RMSE» معمولاً معیار عملکرد ترجیحی برای مسائل رگرسیون است، در برخی موارد ممکن است بخواهید از تابع دیگری استفاده کنید؛ برای مثال، وقتی که «مناطق پَرت» زیادی در داده‌ها وجود دارد. در این شرایط، می‌توان از « میانگین قدر مطلق خطا (MAE: *Mean Absolute Error*)» استفاده کرد، که به آن «میانگین قدر مطلق انحراف (*average absolute deviation*)» نیز گفته می‌شود. فرمول این معیار در معادله ۲-۲ نشان داده شده است:

معادله ۲-۲. میانگین قدر مطلق خطا (MAE)

$$\mathrm{MAE}(\mathrm{X}, \mathrm{h}) = \frac{1}{m}\sum_{i=1}^{m}\left|\left(\mathrm{h}(\mathrm{x}^{(\mathrm{i})}) - \mathrm{y}^{(\mathrm{i})}\right)\right|$$

هم «RMSE» و هم «MAE» روش‌هایی برای اندازه‌گیری فاصله بین دو بردار هستند: «بردار پیش‌بینی‌ها (prediction vector)» و «بردار مقادیر هدف (target vector)». معیارهای مختلف اندازه‌ی فاصله یا «نُرم‌ها (*norms*)»—نمایانگر بزرگی یا فاصله‌ی بردار از مبدأ است—می‌توانند به کار گرفته شوند:

- محاسبه «RMSE» با «نُرم اقلیدسی (*Euclidean norm*)» مطابقت دارد؛ این همان مفهوم فاصله‌ای است که همه ما با آن آشنا هستیم. همچنین به آن «نُرم ℓ_2 (ℓ_2 *norm*)» گفته می‌شود و با $\|\cdot\|_2$ یا گاهی فقط $\|\cdot\|$ نمایش داده می‌شود.

© 2025 Mohsen Zare. All rights reserved.

- محاسبه «MAE» مطابق با «نُرم ℓ_1 (ℓ_1 $norm$)» است و با $\| \cdot \|_1$ نمایش داده می‌شود. این نُرم گاهی «نُرم مَنهَتن ($Manhattan\ norm$)» نامیده می‌شود، زیرا فاصله بین دو نقطه را همانند حرکت در یک شهر فرض می‌کند، جایی که تنها می‌توان در امتداد بلوک‌های متعامد حرکت کرد.

- به‌طور کلی‌تر، «نُرم ℓ_k (ℓ_k $norm$)» یک بردار $\mathbf{v}$ شامل n عنصر به صورت زیر تعریف می‌شود: $\| v \|_k = (|v_1|^k + |v_2|^k + \ldots + |v_n|^k)^{1/k}$ تعریف می‌شود. ℓ_0 تعداد عناصر غیر صفر در بردار و ℓ_∞ حداکثر مقدار مطلق در بردار را نشان می‌دهد. «ℓ_0» تعداد عناصر غیرصفر در بردار را نشان می‌دهد و «ℓ_∞» ماکسیمم مقدار مطلق عناصر بردار را مشخص می‌کند.

هرچه « شاخصِ نُرم ($norm\ index$)» بالاتر باشد، تمرکز بیشتری روی مقادیر بزرگ دارد و مقادیر کوچک‌تر را نادیده می‌گیرد. به همین دلیل، «RMSE» نسبت به «MAE» به «مقادیر پرت» حساس‌تر است. با این حال، وقتی مقادیر پرت به شکل نمایی و به ندرت رخ دهند (مانند یک توزیع «زنگوله‌ای شکل(bell-shaped)»)، RMSE عملکرد بسیار خوبی ارائه می‌دهد و معمولاً ترجیح داده می‌شود.

© 2025 Mohsen Zare. All rights reserved.

بررسی فرضیات

در نهایت، فهرست کردن و تأیید «فرضیات (assumptions)» ای که تاکنون توسط شما یا دیگران انجام شده، تمرین مفیدی است. این کار به شما کمک می‌کند تا مشکلات جدی را در مراحل اولیه شناسایی کنید. به عنوان مثال، «قیمت‌های منطقه‌ای» که سیستم شما تولید می‌کند، به یک «سیستم یادگیری ماشینی پایین‌دستی (downstream machine learning system)» وارد می‌شوند و شما فرض می‌کنید که این قیمت‌ها به همان شکل استفاده خواهند شد. اما اگر سیستم پایین‌دستی، قیمت‌ها را به «دسته‌بندی‌ها (categories)» مانند «ارزان»، «متوسط» یا «گران» تبدیل کند و سپس از این دسته‌بندی‌ها به جای خود قیمت‌ها استفاده کند، چه اتفاقی می‌افتد؟ در این حالت، دقت مقدار واقعی قیمت اهمیت ندارد؛ سیستم شما تنها باید دسته‌بندی را درست انجام دهد. اگر چنین باشد، مسئله باید به عنوان یک «مسئله طبقه‌بندی (classification task)» مطرح می‌شد، نه یک «مسئله رگرسیون (regression task)». شما نمی‌خواهید که این موضوع را، پس از ماه‌ها کار روی یک سیستم رگرسیون متوجه شوید.

خوشبختانه، پس از گفتگو با تیم مسئول «سیستم پایین‌دستی»، مطمئن می‌شوید که آنها واقعاً به «قیمت‌های واقعی» نیاز دارند، نه صرفاً دسته‌بندی‌ها. عالی! همه چیز آماده است، چراغ‌ها سبز شده و می‌توانید اکنون «کدنویسی» را آغاز کنید.

© 2025 Mohsen Zare. All rights reserved.

داده‌ها را دریافت کنید

وقت آن رسیده که وارد عمل شوید. در برداشتن «لپ‌تاپ» خودتان و بررسی «مثال‌های کد (code examples)» تردید نکنید. همان‌طور که در مقدمه اشاره شد، تمام مثال‌های کد در این کتاب «متن‌باز (open-source)» هستند و به صورت آنلاین به عنوان «نوت‌بوک‌های Jupyter» در دسترس قرار دارند؛ این دفترچه‌های یادداشت اسناد تعاملی حاوی متن، تصاویر و قطعه کدهای اجرایی هستند (در موضوع مورد مطالعه‌ی ما، زبان برنامه‌نویسی «پایتون» است). در این کتاب فرض شده است که شما این نوت‌بوک‌ها را در «Google Colab» اجرا می‌کنید، یک سرویس رایگان که امکان اجرای هر نوت‌بوک Jupyter را مستقیماً به صورت آنلاین و بدون نیاز به نصب نرم‌افزار روی دستگاه فراهم می‌کند. اگر ترجیح می‌دهید از پلتفرم آنلاین دیگری مانند «Kaggle» استفاده کنید یا همه چیز را به صورت محلی روی دستگاه خود نصب کنید، لطفاً به صفحهٔ «دستورالعمل‌های GitHub» کتاب مراجعه کنید.

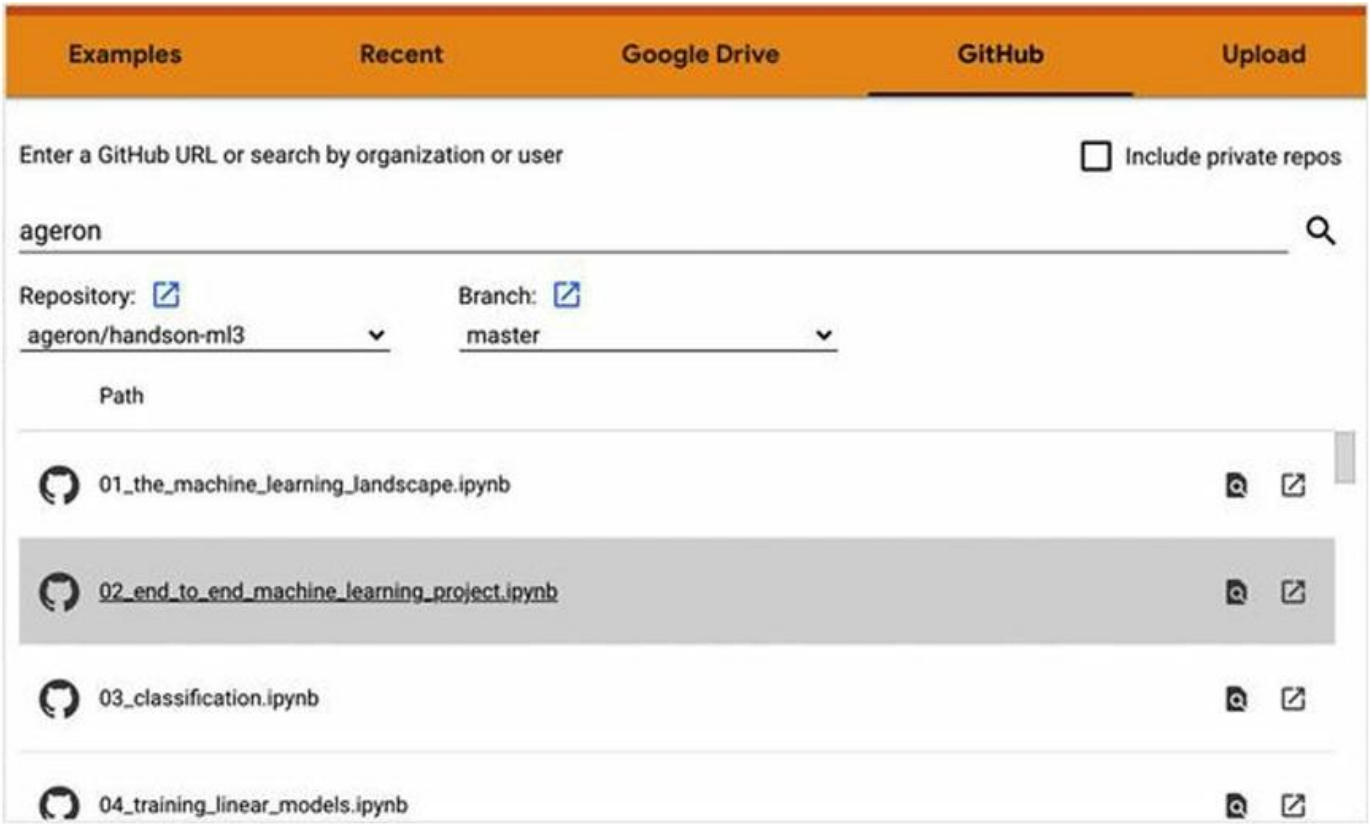
© 2025 Mohsen Zare. All rights reserved.

اجرای مثال‌های کد با استفاده از Google Colab

ابتدا یک مرورگر وب باز کنید و به آدرس *https://homl.info/colab3* مراجعه کنید؛ این لینک شما را به «Google Colab» هدایت می‌کند و فهرست «نوت‌بوک‌های Jupyter» مربوط به این کتاب را نمایش می‌دهد (شکل ۳-۲ را ببینید). در هر فصل، یک نوت‌بوک اصلی به همراه چند نوت‌بوک و آموزش اضافی برای «NumPy» ، «Matplotlib» ، «Pandas» ، «جبر خطی (linear algebra)» و «حساب دیفرانسیل و انتگرال (calculus)» خواهید یافت. به عنوان مثال، اگر روی «*02_end_to_end_machine_learning_project.ipynb*» کلیک کنید، نوت‌بوک فصل ۲ در Google Colab باز خواهد شد (شکل ۴-۲ را ببینید).

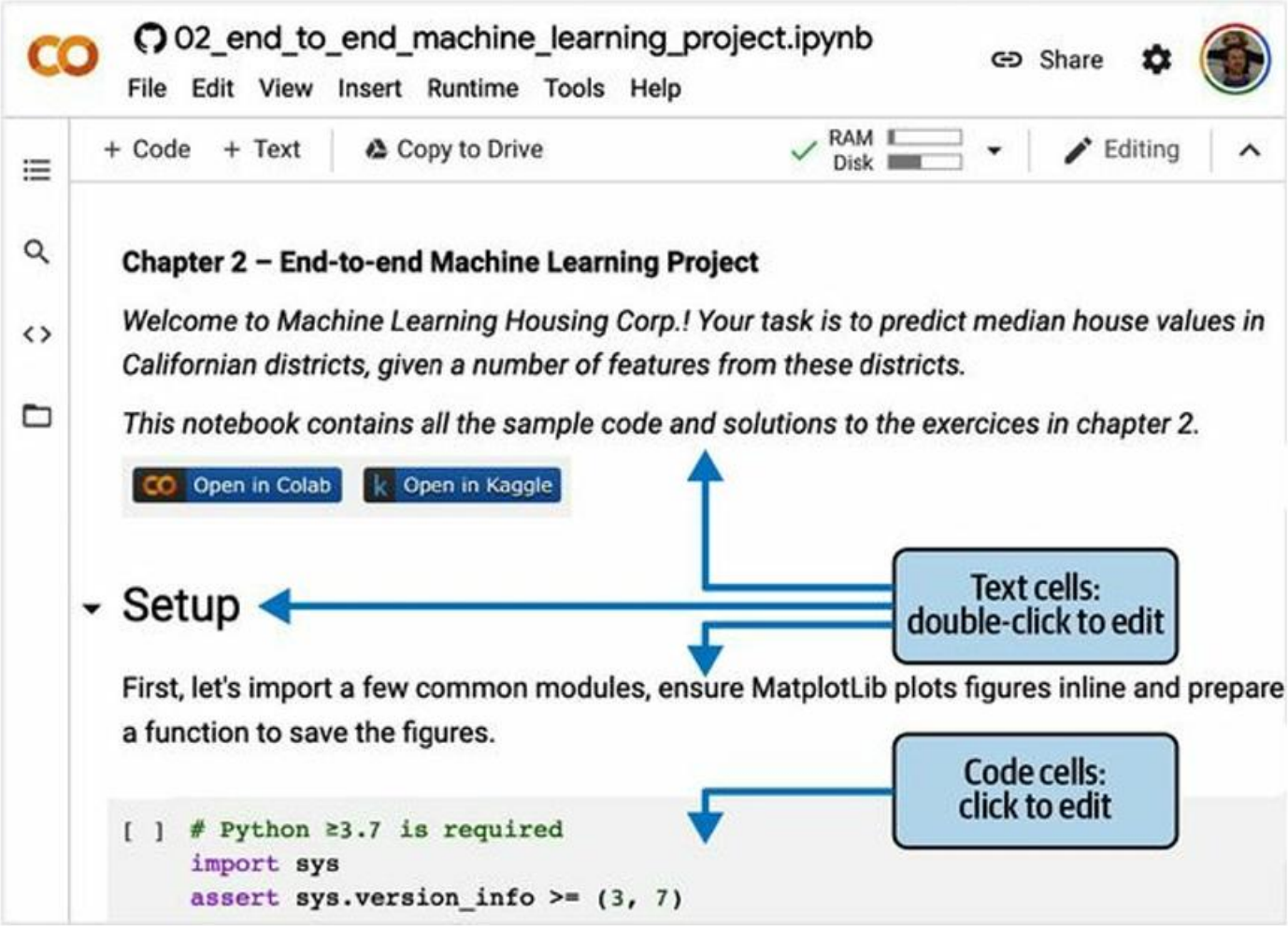
یک «نوت‌بوک Jupyter» از فهرستی از «سلول‌ها (cells)» تشکیل شده و هر سلول شامل «کد اجرایی (executable code)» یا متن است. روی اولین سلول متنی (که شامل جمله «به شرکت مسکن یادگیری ماشینی خوش آمدید!» است) دوبار کلیک کنید تا سلول برای «ویرایش (editing)» باز شود. توجه داشته باشید که نوت‌بوک‌های Jupyter از «قواعد نوشتاری(Markdown syntax)» برای قالب‌بندی استفاده می‌کنند، مانند **bold** ، *italics* ، # Title، [link text](url) و غیره. سعی کنید متن را تغییر دهید و سپس کلیدهای Shift+Enter را فشار دهید تا نتیجه را مشاهده کنید.

© 2025 Mohsen Zare. All rights reserved.



شکل ۲-۳. فهرست نوت‌بوک‌ها در Google Colab

© 2025 Mohsen Zare. All rights reserved.



شکل ۲-۴. نوت‌بوک شما در Google Colab

سپس، از منو «Insert → Code cell» یک «code cell» جدید ایجاد کنید. روش دیگر این است که روی دکمه Code+ در «نوار ابزار(toolbar)» کلیک کنید، یا موس خود را پایین یک سلول نگه دارید تا گزینه‌های Code+ و Text+ ظاهر شوند و سپس روی Code+ کلیک کنید. در سلول کد جدید، مقداری کد پایتون، مانند print("Hello World")، تایپ کنید، سپس Shift+Enter را فشار دهید تا این کد اجرا شود (یا روی دکمه ▷ در سمت چپ سلول کلیک کنید).

اگر به «حساب گوگل (Google account)» خود وارد نشده‌اید، از شما خواسته می‌شود اکنون وارد شوید. (اگر قبلاً حساب گوگل ندارید، باید یکی ایجاد کنید.) پس از ورود، هنگام تلاش برای اجرای «کد»، ممکن است یک هشدار امنیتی مشاهده کنید که اطلاع می‌دهد این «notebook» توسط گوگل ایجاد نشده است. یک شخص خرابکار می‌تواند نوت‌بوک‌ای طراحی کند که سعی کند شما را فریب دهد تا اطلاعات کاربری گوگل خود را وارد کنید و به اطلاعات شخصی‌تان دسترسی پیدا کند. بنابراین، قبل از

© 2025 Mohsen Zare. All rights reserved.

اجرای هر نوت‌بوک، همیشه مطمئن شوید که به نویسنده اعتماد دارید، یا عملکرد هر سلول کد را بررسی کرده‌اید. با فرض اینکه به نویسنده اعتماد دارید یا قصد دارید هر سلول کد را بررسی کنید، اکنون می‌توانید روی «Run anyway» کلیک کنید.

پس از این مرحله «Colab» یک «محیط اجرا (*runtime*)» جدید—منظور همان محیط پردازشی (CPU/GPU/TPU) است که کد در داخل آن اجرا می‌شود—برای شما اختصاص می‌دهد. این محیط یک «ماشین مجازی (*virtual machine*)» رایگان است که روی سرورهای گوگل قرار دارد و شامل مجموعه‌ای از ابزارها و «کتابخانه‌های پایتون (Python libraries)» است، از جمله تمام مواردی که برای اکثر فصل‌ها نیاز دارید. (در برخی فصل‌ها ممکن است لازم باشد یک کتابخانه اضافی را با یک دستور نصب کنید.) این فرایند چند ثانیه طول می‌کشد. سپس Colab به طور خودکار به محیط اجرا متصل می‌شود و از آن برای اجرای «سلول‌های کد» شما استفاده می‌کند. مهم‌تر اینکه، کد در این محیط اجرا می‌شود، «نه» روی دستگاه شما، و خروجی آن مستقیماً زیر سلول نمایش داده می‌شود.

تبریک می‌گویم! شما موفق شدید چند خط کد پایتون را در Colab اجرا کنید.

نکته

برای افزودن یک «سلول کد» جدید، می‌توانید Ctrl-M (یا Cmd-M در macOS) و سپس A (برای درج سلول بالای سلول فعال) یا B (برای درج زیر سلول فعال) را فشار دهید. میانبرهای صفحه‌کلید دیگری نیز وجود دارند. اگر Ctrl-M (یا Cmd-M) و سپس H را تایپ کنید، همه میانبرها را مشاهده و در صورت نیاز می‌توانید آن‌ها را ویرایش کنید. اگر تصمیم دارید نوت‌بوک‌ها را در Kaggle یا روی دستگاه خود با JupyterLab یا یک IDE مانند Visual Studio Code با افزونه Jupyter اجرا کنید، ممکن است تفاوت‌های جزئی ببینید: ماشین‌های مجازی در آنجا «هسته پردازشی (*kernel*)» نامیده می‌شوند، رابط کاربری کمی متفاوت است و میانبرهای صفحه‌کلید نیز فرق دارند. با این حال، جابجایی بین محیط‌های Jupyter خیلی سخت نیست.

© 2025 Mohsen Zare. All rights reserved.

ذخیره تغییرات کد و داده‌ها

شما می‌توانید تغییرات خود را در یک «نوت‌بوکِ کولَب (Colab notebook)» ایجاد کنید و این تغییرات تا زمانی که تب مرورگر خود را باز نگه دارید، حفظ می‌شوند. اما زمانیکه تب را ببندید، تغییرات از بین می‌روند. برای جلوگیری از این مشکل، مطمئن شوید که یک نسخه از نوت‌بوک‌تان را در «Google Drive» خود ذخیره می‌کنید: از منوی File → Save a copy in Drive استفاده کنید. همچنین می‌توانید نوت‌بوک را روی کامپیوتر خود دانلود کنید: از مِنوی File → Download → Download .ipynb استفاده کنید. بعداً می‌توانید دوباره به آدرس *https://colab.research.google.com* بروید و نوت‌بوک را باز کنید(یا از Google Drive یا از فایل دانلود شده روی کامپیوتر خود.)

هشدار

« گوگل کولب» فقط برای استفاده تعاملی در نظر گرفته شده است: شما می‌توانید در نوت‌بوک‌ها بازی کنید و کد را به دلخواه تغییر دهید، اما نمی‌توانید اجازه دهید نوت‌بوک‌ها برای مدت طولانی بدون نظارت اجرا شوند. در غیر این صورت، «محیط اجرا» خاموش می‌شود و تمام داده‌های موجود در آن از بین می‌رود.

اگر نوت‌بوک Jupyter داده‌هایی تولید می‌کند که برای شما مهم هستند، حتماً قبل از خاموش شدن محیط اجرایی آن‌ها را دانلود کنید. برای این کار:

۱. روی آیکون «Files» کلیک کنید (به مرحله ۱ در شکل ۲-۵ مراجعه کنید).
۲. فایل مورد نظر را پیدا کرده و روی سه نقطه عمودی کنار آن کلیک کنید (مرحله ۲).
۳. گزینه «Download» را انتخاب کنید (مرحله ۳).

© 2025 Mohsen Zare. All rights reserved.

به عنوان روش جایگزین، می‌توانید «گوگل درایو (Google Drive)» خود را به محیط اجرا متصل کنید تا نوت‌بوک بتواند فایل‌ها را مستقیماً بخواند و ذخیره کند، گویی که یک «دایرکتوری محلی (local directory)» است. برای این کار:

۱. روی آیکون «Files» کلیک کنید (مرحله ۱).

۲. سپس روی آیکون «Google Drive» (دایره شده در شکل ۲-۵) کلیک کنید و دستورالعمل‌های نمایش داده شده را دنبال کنید.

شکل ۲-۵. دانلود یک فایل از یک محیط اجرا Google Colab (مراحل ۱ تا ۳) یا نصب Google Drive خودتان (آیکون دایره‌ای)

به طور پیش‌فرض، گوگل درایو شما در مسیر content/drive/MyDrive/ قرار می‌گیرد. اگر بخواهید از یک فایل داده نسخه پشتیبان تهیه کنید، کافی است آن را با اجرای دستور *!cp/content/my_great_model/content/drive/MyDrive* در این مسیر کپی کنید. هر دستوری که با علامت تعجب (!) یا bang شروع شود، به عنوان دستور shell و نه کد پایتون، اجرا می‌شود. دستور cp در لینوکس برای کپی کردن فایل‌ها از یک مسیر به مسیر دیگر استفاده می‌شود. توجه داشته باشید که محیط‌های اجرایی Colab روی لینوکس، به طور خاص اوبونتو، اجرا می‌شوند.

© 2025 Mohsen Zare. All rights reserved.

قدرت و خطر تعامل

نوت‌بوک‌های Jupyter ماهیتی تعاملی دارند و این ویژگی یک مزیت بزرگ محسوب می‌شود: شما می‌توانید هر سلول را به‌صورت جداگانه اجرا کنید، در هر نقطه‌ای متوقف شوید، سلول‌های جدید اضافه کنید، با کد آزمایش کنید، تغییرات را اعمال کرده و همان سلول را دوباره اجرا کنید. من اکیداً توصیه می‌کنم این تعامل را جدی بگیرید؛ اجرای صرف سلول‌ها بدون آزمایش و دستکاری آنها سرعت یادگیری شما را کاهش می‌دهد. با این حال، این انعطاف‌پذیری خطراتی نیز دارد: اجرای سلول‌ها به ترتیب اشتباه یا فراموش کردن اجرای یک سلول، اتفاقی ساده و محتمل است که می‌تواند باعث از کار افتادن سلول‌های بعدی شود. برای مثال، اولین سلول کد در هر نوت‌بوک معمولاً شامل کد راه‌اندازی است، مانند تعریف ورودی‌ها؛ بنابراین حتماً ابتدا آن را اجرا کنید، در غیر این صورت بخش‌های بعدی به درستی عمل نخواهند کرد.

نکته

اگر با خطای غیرمنتظره‌ای مواجه شدید، ابتدا محیط اجرا را مجدداً راه‌اندازی کنید (از منوی "Runtime" گزینه "Restart runtime" را انتخاب کنید) و سپس تمام سلول‌ها را دوباره از ابتدای نوت‌بوک اجرا کنید. اغلب این کار مشکل را برطرف می‌کند. در صورتی که مشکل همچنان باقی بماند، احتمالاً یکی از تغییراتی که اعمال کرده‌اید باعث اختلال در نوت‌بوک شده است؛ در این صورت به نسخه اصلی نوت‌بوک بازگردید و دوباره امتحان کنید. اگر باز هم مشکل حل نشد، لطفاً موضوع را در GitHub ثبت کنید تا بررسی شود.

© 2025 Mohsen Zare. All rights reserved.

کد کتاب در مقابل کد نوت‌بوک

گاهی ممکن است متوجه تفاوت‌های جزئی بین کدهای موجود در این کتاب و کدهای نوت‌بوک شوید. این اختلافات می‌تواند دلایل مختلفی داشته باشد:

- ممکن است هنگام مطالعه شما، برخی کتابخانه‌ها تغییر کرده باشند یا با وجود تمام تلاش‌ها، خطایی در نسخه چاپی کتاب وجود داشته باشد. متأسفانه نمی‌توانم کد موجود در نسخه چاپی شما را به‌طور خودکار اصلاح کنم (مگر اینکه نسخه الکترونیکی کتاب را داشته باشید و بتوانید آخرین نسخه را دانلود کنید)، اما نوت‌بوک‌ها به‌روز هستند. بنابراین اگر پس از کپی کردن کد از کتاب با خطا مواجه شدید، لطفاً نسخه اصلاح شده کد را در نوت‌بوک‌ها بررسی کنید. من تمام تلاشم را می‌کنم تا نوت‌بوک‌ها همیشه بدون خطا و با آخرین نسخه‌های کتابخانه‌ها به‌روز باشند.

- نوت‌بوک‌ها همچنین شامل کدهای اضافی برای بهبود ظاهر شکل‌ها (افزودن برچسب‌ها، تنظیم اندازه فونت و غیره) و ذخیره آن‌ها با کیفیت بالا برای این کتاب هستند. در صورت تمایل، می‌توانید این کدهای اضافی را نادیده بگیرید.

من کد را برای خوانایی و سادگی بهینه کرده‌ام: ساختار آن تا حد ممکن خطی و مسطح است و فقط تعداد کمی تابع یا کلاس تعریف شده است. هدف این است که مطمئن شویم کدی که اجرا می‌کنید، به‌طور مستقیم و واضح قابل مشاهده است و نیازی به جستجو در چندین لایه انتزاعی ندارد. این طراحی همچنین باعث می‌شود بازی با کد و آزمایش آن برای شما آسان‌تر باشد. برای ساده‌تر کردن کد، مدیریت خطا محدود است و برخی از importهای کم‌استفاده دقیقاً در همان نقطه‌ای قرار گرفته‌اند که مورد نیاز هستند، نه در بالای فایل، همان‌طور که در راهنمای سبک PEP 8 Python توصیه شده است. با این وجود، کد تولیدی شما تفاوت زیادی نخواهد داشت؛ فقط کمی ماژولارتر است و شامل آزمون‌ها و مدیریت خطاهای اضافی می‌شود.

عالی! وقتی کار با Colab برای شما راحت شد، آنوقت آماده‌اید که داده‌ها را دانلود کنید.

© 2025 Mohsen Zare. All rights reserved.

دانلود داده‌ها

در محیط‌های معمول، داده‌های شما اغلب در یک پایگاه داده رابطه‌ای یا سایر مخازن داده رایج ذخیره شده و در چندین جدول، سند یا فایل پراکنده‌اند. برای دسترسی به آن‌ها، ابتدا باید اعتبارنامه‌ها و مجوزهای لازم را دریافت کنید و با ساختار داده‌ها (data schema) آشنا شوید. با این حال، در این پروژه همه چیز ساده‌تر است: شما تنها یک فایل فشرده به نام *housing.tgz* دانلود می‌کنید که شامل یک فایل CSV به نام *housing.csv* است و تمامی داده‌های مورد نیاز در آن قرار دارند.

به جای دانلود و خارج کردن دستی داده‌ها از حالت فشرده، معمولاً ترجیح داده می‌شود تابعی بنویسید که این کار را برای شما انجام دهد. این امر به ویژه زمانی مفید است که داده‌ها مرتباً تغییر کنند: می‌توانید یک اسکریپت کوچک بنویسید که از این تابع برای دریافت آخرین داده‌ها استفاده کند (یا می‌توانید یک کار زمان‌بندی شده تنظیم کنید تا آن را به طور خودکار در فواصل منظم انجام دهد). خودکارسازی فرآیند دریافت داده‌ها در صورتی که نیاز به نصب مجموعه داده‌ها روی چندین دستگاه داشته باشید نیز مفید است.

این تابع برای دریافت و بارگذاری داده‌ها است:

```
from pathlib import Path
import pandas as pd
import tarfile
import urllib.request

def load_housing_data():
    tarball_path = Path("datasets/housing.tgz")
    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://github.com/ageron/data/raw/main/housing.tgz"
        urllib.request.urlretrieve(url, tarball_path)
        with tarfile.open(tarball_path) as housing_tarball:
            housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()
```

© 2025 Mohsen Zare. All rights reserved.

هنگامی که تابع load_housing_data() فراخوانی می‌شود، ابتدا به دنبال فایل *datasets/housing.tgz* می‌گردد. اگر این فایل موجود نباشد، تابع ابتدا دایرکتوری *datasets* را در مسیر فعلی ایجاد می‌کند (در Colab مسیر پیش‌فرض content/ است)، سپس فایل *housing.tgz* را از مخزن ageron/data در GitHub دانلود کرده و محتوای آن را در همان دایرکتوری استخراج می‌کند. این عملیات باعث ایجاد مسیر datasets/housing و فایل housing.csv درون آن می‌شود. در نهایت، تابع فایل CSV را در یک آبجکت Pandas DataFrame بارگذاری می‌کند و آن را برمی‌گرداند تا شامل تمامی داده‌ها باشد.

© 2025 Mohsen Zare. All rights reserved.

نگاهی سریع به ساختار داده

شما کار خود را با مشاهده پنج ردیف اول داده‌ها آغاز می‌کنید و برای این کار از متد head() در DataFrame استفاده می‌کنید (شکل ۲-۶ را ببینید).

```
housing.head()
```

	longitude	latitude	housing_median_age	median_income	ocean_proximity	median_house_value
0	-122.23	37.88	41.0	8.3252	NEAR BAY	452600.0
1	-122.22	37.86	21.0	8.3014	NEAR BAY	358500.0
2	-122.24	37.85	52.0	7.2574	NEAR BAY	352100.0
3	-122.25	37.85	52.0	5.6431	NEAR BAY	341300.0
4	-122.25	37.85	52.0	3.8462	NEAR BAY	342200.0

شکل ۲-۶. پنج ردیف بالایی در مجموعه داده‌ها

هر ردیف نشان‌دهنده یک منطقه است و داده‌ها شامل ۱۰ ویژگی می‌شوند (همه ویژگی‌ها در تصویر نمایش داده نشده‌اند): طول جغرافیایی، عرض جغرافیایی، میانگین سن مسکن، تعداد کل اتاق‌ها، تعداد کل اتاق‌خواب‌ها، جمعیت، تعداد خانوارها، میانگین درآمد، میانگین ارزش خانه و نزدیکی به اقیانوس.

متد info() برای به‌دست آوردن یک شرح یا نمای کلیِ سریع از داده‌ها بسیار مفید است، به‌ویژه برای مشاهده تعداد کل ردیف‌ها، نوع داده هر ویژگی و تعداد مقادیر غیر خالی یا معتبر در هر ستون.

```
>>> housing.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
#   Column          Non-Null Count  Dtype
---  ------          --------------  -----
0   longitude       20640 non-null  float64
1   latitude        20640 non-null  float64
2   housing_median_age    20640 non-null  float64
3   total_rooms     20640 non-null  float64
```

© 2025 Mohsen Zare. All rights reserved.

```
4   total_bedrooms   20433 non-null  float64
5   population       20640 non-null  float64
6   households       20640 non-null  float64
7   median_income    20640 non-null  float64
8   median_house_value   20640 non-null  float64
9   ocean_proximity  20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

نکته

در این کتاب، هنگامی که یک مثال کد شامل ترکیبی از کد و خروجی‌ها باشد، برای خوانایی بهتر، همانند مفسر پایتون قالب‌بندی می‌شود: خطوط کد با پیشوند <<< (و برای بلوک‌های تودرتو با ...) شروع می‌شوند، در حالی که خروجی‌ها بدون پیشوند نمایش داده می‌شوند.

مجموعه‌داده شامل ۲۰,۶۴۰ نمونه است؛ این اندازه نسبتاً کوچک به شمار می‌آید، اما برای شروع یادگیری ماشین مناسب است. توجه می‌کنید که ویژگی total_bedrooms تنها ۲۰,۴۳۳ مقدار غیر تهی دارد، یعنی ۲۰۷ منطقه فاقد این ویژگی هستند و باید بعداً این کمبود داده را مدیریت کنید.

تمام ویژگی‌ها عددی هستند، به جز ocean_proximity. نوع آن object است و بنابراین می‌تواند هر نوع شیء پایتون را در خود جای دهد. با این حال، چون داده‌ها را از یک فایل CSV بارگذاری کرده‌اید، می‌دانید که این ویژگی متنی است. با نگاه کردن به پنج ردیف اول، احتمالاً متوجه شده‌اید که مقادیر ستون ocean_proximity تکراری هستند، یعنی این ویژگی یک ویژگیِ دسته‌بندی محسوب می‌شود. با استفاده از متد value_counts() می‌توانید ببینید چه دسته‌هایی وجود دارند و هر دسته شامل چند ناحیه است:

```
>>> housing["ocean_proximity"].value_counts()
<1H OCEAN     9136
INLAND        6551
```

© 2025 Mohsen Zare. All rights reserved.

```
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

بیایید نگاهی به سایر فیلدها بیندازیم. متد ()describe یک خلاصه آماری از ویژگی‌های عددی ارائه می‌دهد و اطلاعاتی مانند میانگین، انحراف معیار، مینیمم و ماکسیمم مقادیر و چارک‌ها را نمایش می‌دهد (شکل ۲-۷).

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	500001.000000

شکل ۲-۷. خلاصه هر ویژگی عددی

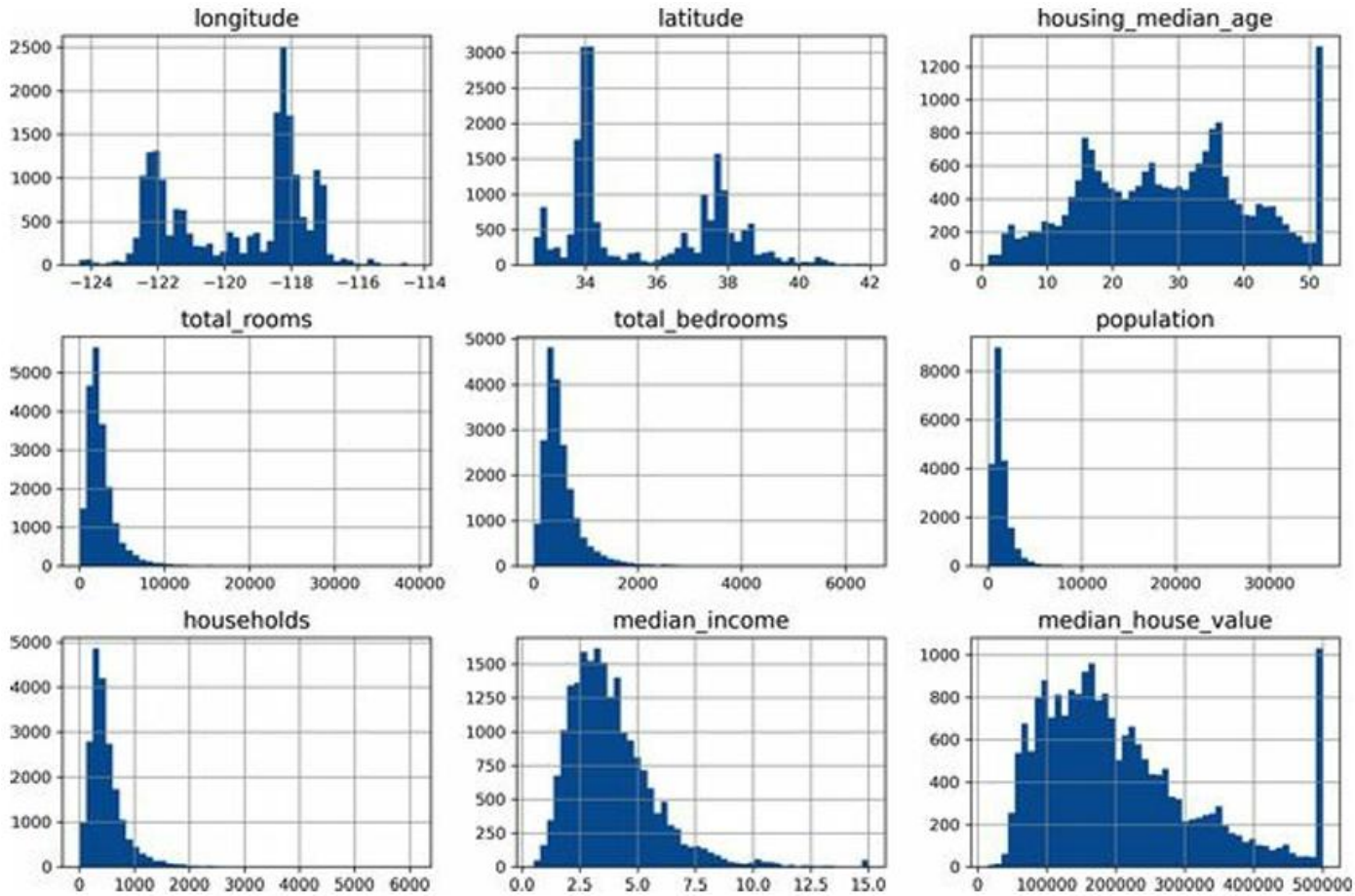
ردیف‌های count، mean، min و max نیاز به توضیح ندارند. توجه داشته باشید که مقادیر null نادیده گرفته می‌شوند؛ بنابراین، برای مثال، count مربوط به total_bedrooms برابر با ۲۰,۴۳۳ است، نه ۲۰,۶۴۰. ردیف std نشان‌دهنده «انحراف معیار (*standard deviation*)» است که میزان پراکندگی مقادیر را اندازه‌گیری می‌کند. ردیف‌های ۲۵%، ۵۰% و ۷۵% «صدک‌ها (*percentiles*)»ی مربوطه را نشان می‌دهند؛ صدک مقداری است که درصد مشخصی از مشاهدات در یک گروه از آن پایین‌تر قرار می‌گیرند. به عنوان مثال، ۲۵% از مناطق دارای میانگین سن مسکن کمتر از ۱۸ هستند، ۵۰% کمتر از ۲۹ و ۷۵% کمتر از ۳۷. این موارد معمولاً به نام صدک ۲۵ (یا چارک اول)، صدک ۵۰ یا میانه و صدک ۷۵ (یا چارک سوم) شناخته می‌شوند.

یک راه سریع دیگر برای درک نوع داده‌هایی که با آنها سر و کار دارید، رسم یک هیستوگرام برای هر ویژگی عددی است. یک هیستوگرام تعداد نمونه‌هایی (روی محور عمودی) را نشان می‌دهد که دارای

© 2025 Mohsen Zare. All rights reserved.

محدوده مقدار مشخصی (روی محور افقی) هستند. می‌توانید این ویژگی را به صورت تک تک رسم کنید، یا می‌توانید متد hist() را روی کل مجموعه داده‌ها فراخوانی کنید (همانطور که در مثال کد زیر نشان داده شده است)، و این متد برای هر ویژگی عددی یک هیستوگرام رسم می‌کند (شکل ۲-۸ را ببینید):

```
import matplotlib.pyplot as plt

housing.hist(bins=50, figsize=(12, 8))

plt.show()
```



شکل ۲-۸. یک هیستوگرام برای هر ویژگی عددی

با بررسی این هیستوگرام‌ها، چند نکته جلب توجه می‌کند:

© 2025 Mohsen Zare. All rights reserved.

- نخست، به نظر نمی‌رسد ویژگی میانگین درآمد به دلار آمریکا (USD) بیان شده باشد. پس از بررسی با تیم گردآورنده داده‌ها، مشخص می‌شود که مقادیر برای درآمدهای متوسط بالا در محدوده حداکثر ۱۵ (در واقع، ۱۵/۰۰۰۱) و برای درآمدهای متوسط پایین در حداقل ۰/۵ (در واقع، ۰/۴۹۹۹) محدود شده‌اند. این مقیاس‌بندی تقریباً معادل ده‌ها هزار دلار است (برای مثال، مقدار ۳ به معنای حدود ۳۰,۰۰۰ دلار است). کار با ویژگی‌هایی که از پیش پردازش شده‌اند، در یادگیری ماشین امری رایج است و لزوماً مشکل‌ساز نیست، اما بهتر است نحوه محاسبه و پردازش این داده‌ها را به‌طور دقیق بدانید.

- میانگین سنی مسکن و میانگین ارزش خانه نیز دارای محدودیت مقادیر هستند. محدودیت در ویژگی دوم می‌تواند مشکل‌ساز باشد، زیرا این ویژگی همان متغیر هدف (برچسب‌ها) شما است. در این حالت، الگوریتم‌های یادگیری ماشین ممکن است یاد بگیرند که قیمت‌ها هرگز از آن حد فراتر نمی‌روند. بنابراین لازم است با تیم مشتری (تیمی که قرار است از خروجی سیستم شما استفاده کند) بررسی کنید که آیا این موضوع یک مشکل محسوب می‌شود یا خیر. اگر مشخص شد که آن‌ها حتی برای مقادیر بالاتر از ۵۰۰,۰۰۰ دلار به پیش‌بینی دقیق نیاز دارند، دو راه‌حل وجود دارد:
  - جمع‌آوری برچسب‌های واقعی برای مناطقی که مقدار برچسب آن‌ها محدود شده است.
  - حذف این مناطق از مجموعه آموزش و همچنین مجموعه آزمون، تا در صورتی که سیستم مقادیری بالاتر از ۵۰۰,۰۰۰ دلار پیش‌بینی کند، ارزیابی آن به‌صورت نادرست پایین در نظر گرفته نشود

- این ویژگی‌ها دارای مقیاس‌های بسیار متفاوتی هستند. در ادامه این فصل، هنگام بررسی مبحث مقیاس‌بندی ویژگی‌ها، به این موضوع پرداخته خواهد شد.

- همچنین، بسیاری از هیستوگرام‌ها «دارای توزیع نامتوازن متمایل به راست (*skewed right*)»—یک نوع توزیع نامتقارنی که در آن بیشتر داده‌ها کوچک هستند و دم طولانی به سمت مقادیر بزرگ‌تر کشیده شده، و می‌تواند بر میانگین و واریانس اثر بگذارد—هستند؛ به این معنا که بخش زیادی از داده‌ها در سمت چپ متمرکز شده و دنباله آن‌ها به سمت راست کشیده شده است. این حالت ممکن است شناسایی الگوها را برای برخی

© 2025 Mohsen Zare. All rights reserved.

الگوریتم‌های یادگیری ماشین دشوارتر کند. در مراحل بعد، سعی خواهید کرد این ویژگی‌ها را طوری تبدیل کنید که توزیع آن‌ها متقارن‌تر و نزدیک به شکل زنگوله‌ای باشد.

اکنون دید بهتری نسبت به نوع داده‌هایی که با آن‌ها کار می‌کنید، خواهید داشت.

هشدار

پیش از آنکه بررسی عمیق‌تری روی داده‌ها انجام دهید، لازم است یک مجموعه آزمایشی ایجاد کرده و آن را کنار بگذارید. این مجموعه باید تا پایان فرآیند توسعه مدل بدون استفاده باقی بماند و هیچ‌گونه تغییر یا بررسی مستقیمی روی آن انجام نشود. هدف از این کار، ارزیابی بی‌طرفانه عملکرد نهایی مدل بر روی داده‌هایی است که هرگز در فرآیند آموزش و تنظیم مدل دیده نشده‌اند.

© 2025 Mohsen Zare. All rights reserved.

ایجاد یک مجموعه‌ی آزمون

شاید در این مرحله کنار گذاشتن بخشی از داده‌ها برای استفاده به‌عنوان مجموعه آزمون کمی عجیب به نظر برسد. در واقع، شما تازه بررسی اولیه داده‌ها را آغاز کرده‌اید و منطقی است که بخواهید قبل از انتخاب الگوریتم مناسب، اطلاعات بیشتری از داده‌ها به‌دست آورید. این درست است، اما باید به یاد داشت که مغز انسان یک سیستم فوق‌العاده برای تشخیص الگوها است و همین ویژگی می‌تواند باعث بروز بیش‌برازش شود.

اگر شما به داده‌های مجموعه آزمون نگاه کنید، ممکن است با الگوهایی مواجه شوید که به نظر جالب می‌رسند و همین امر شما را به انتخاب یک مدل خاص سوق دهد. در نتیجه، وقتی بعداً خطای تعمیم مدل را با استفاده از همین مجموعه آزمون محاسبه می‌کنید، برآورد شما به شکل غیرواقعی خوش‌بینانه خواهد بود. این مسئله می‌تواند منجر به ایجاد سیستمی شود که عملکرد واقعی آن پایین‌تر از حد انتظار است. این پدیده در یادگیری ماشین با نام «بایاسِ تَجسس داده (*data snooping bias*)»–خطا یا انحراف آماری ناشی از وارسی مکرر داده‌ها–شناخته می‌شود.

ایجاد یک مجموعه آزمون از نظر تئوری فرایند ساده‌ای است. کافی است تعدادی نمونه را به‌طور تصادفی از کل مجموعه داده انتخاب کنید—معمولاً حدود ۲۰% از داده‌ها (یا درصدی کمتر در صورتی که مجموعه داده بسیار بزرگ باشد)—و آن‌ها را جدا کرده و برای مراحل بعدی تحلیل یا آموزش به هیچ عنوان استفاده نکنید.

```
import numpy as np

def shuffle_and_split_data(data, test_ratio):
  shuffled_indices = np.random.permutation(len(data))
  test_set_size = int(len(data) * test_ratio)
  test_indices = shuffled_indices[:test_set_size]
  train_indices = shuffled_indices[test_set_size:]
  return data.iloc[train_indices], data.iloc[test_indices]
```

سپس می‌توانید این تابع را به شکل زیر مورد استفاده قرار دهید:

```
>>> train_set, test_set = shuffle_and_split_data(housing, 0.2)
```

© 2025 Mohsen Zare. All rights reserved.

```
>>> len(train_set)
16512
>>> len(test_set)
4128
```

این روش کار می‌کند، اما بی‌عیب نیست: اگر برنامه را دوباره اجرا کنید، مجموعه آزمون متفاوتی به‌دست می‌آید. در طول زمان، شما (یا الگوریتم‌های یادگیری ماشین) ممکن است به کل مجموعه داده دسترسی پیدا کنید، موضوعی که باید از آن جلوگیری شود.

یک راه‌حل این است که مجموعه آزمون را در اولین اجرا ذخیره کرده و در اجرای‌های بعدی از همان نسخه ذخیره‌شده استفاده کنید. راه‌حل دیگر این است که پیش از فراخوانی np.random.permutation() مقدار اولیه مولد اعداد تصادفی را تعیین کنید (برای مثال با np.random.seed(42)) تا همیشه همان شاخص‌های درهم‌ریخته ایجاد شود.

با این حال، هر دو راه‌حل یادشده در زمانی که یک نسخه به‌روزشده از مجموعه داده دریافت کنید، با مشکل روبه‌رو می‌شوند. برای داشتن یک تقسیم‌بندی پایدار آموزش-آزمون (train/test split) حتی پس از به‌روزرسانی داده‌ها، یک روش متداول این است که از شناسه‌ی هر نمونه برای تصمیم‌گیری درباره قرار گرفتن یا نگرفتن آن در مجموعه آزمون استفاده کنید (به شرط آنکه هر نمونه دارای یک شناسه یکتا و تغییرناپذیر باشد).

برای نمونه، می‌توانید یک مقدارِ hash (نگاشت فشرده‌ی داده؛ فرآیند تبدیل داده‌ها به یک مقدار عددی با طول ثابت (hash value) برای تسهیل جستجو، ذخیره‌سازی سریع یا مقایسه داده‌ها) برای شناسه هر نمونه محاسبه کرده و اگر این hash کمتر یا مساوی ۲۰% از بیشترین مقدار hash ممکن باشد، آن نمونه را در مجموعه آزمون قرار دهید. این کار تضمین می‌کند که مجموعه آزمون در تمام اجراها ثابت بماند، حتی اگر داده‌ها به‌روزرسانی شوند. به این ترتیب، نمونه‌های جدید به‌طور تقریبی ۲۰% وارد مجموعه آزمون می‌شوند، اما هیچ‌کدام از نمونه‌هایی که پیش‌تر در مجموعه آموزش بوده‌اند، وارد مجموعه آزمون نخواهند شد.

در اینجا یک نمونه پیاده‌سازی احتمالی ارائه شده است:

```
from zlib import crc32
```

© 2025 Mohsen Zare. All rights reserved.

```
def is_id_in_test_set(identifier, test_ratio):
  return crc32(np.int64(identifier)) < test_ratio * 2**32

def split_data_with_id_hash(data, test_ratio, id_column):
  ids = data[id_column]
  in_test_set = ids.apply(lambda id_: is_id_in_test_set(id_, test_ratio))
  return data.loc[~in_test_set], data.loc[in_test_set]
```

متأسفانه، مجموعه داده‌های مسکن ستونی برای شناسه ندارد. ساده‌ترین راه این است که از شماره ردیف یا اندیس (index) هر ردیف به‌عنوان شناسه استفاده کنید.

```
housing_with_id = housing.reset_index() # adds an `index` column
train_set, test_set = split_data_with_id_hash(housing_with_id, 0.2, "index")
```

اگر از اندیس ردیف به‌عنوان شناسه‌ی منحصر به فرد استفاده می‌کنید، باید اطمینان حاصل کنید که داده‌های جدید به انتهای مجموعه اضافه می‌شوند و هیچ ردیفی حذف نمی‌گردد. اگر این امکان وجود نداشته باشد، می‌توانید از ویژگی‌های پایدارتر برای ساخت یک شناسه‌ی منحصر به فرد استفاده کنید. به عنوان مثال، طول و عرض جغرافیایی یک منطقه در طول میلیون‌ها سال ثابت می‌مانند و می‌توان آن‌ها را برای ایجاد شناسه‌ای مانند نمونه زیر ترکیب کرد:

```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_data_with_id_hash(housing_with_id, 0.2, "id")
```

کتابخانه Scikit-Learn چندین تابع برای تقسیم مجموعه داده‌ها به زیرمجموعه‌های مختلف ارائه می‌دهد. ساده‌ترین آن‌ها تابع train_test_split() است که عملکردی مشابه تابع shuffle_and_split_data() که قبلاً تعریف کردیم، دارد، اما با امکانات اضافی. نخست، پارامتر random_state به شما امکان می‌دهد seed مولد تصادفی را مشخص کنید تا تقسیم‌بندی تکرارپذیر باشد. دوم، می‌توانید چندین مجموعه داده با تعداد ردیف یکسان را به آن ارسال کنید و تابع بر اساس شاخص‌های یکسان آن‌ها را تقسیم می‌کند. این ویژگی به ویژه زمانی مفید است که، به عنوان مثال، برچسب‌ها در یک DataFrame جداگانه ذخیره شده باشند.

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

© 2025 Mohsen Zare. All rights reserved.

تا اینجا روش‌های نمونه‌گیری کاملاً تصادفی را بررسی کرده‌ایم. اگر مجموعه داده شما به اندازه کافی بزرگ باشد، به‌ویژه نسبت به تعداد ویژگی‌ها، این روش معمولاً کارآمد است. اما اگر مجموعه داده کوچک باشد، احتمال ایجاد بایاس در نمونه‌گیری افزایش می‌یابد. به عنوان مثال، وقتی کارکنان یک شرکت نظرسنجی تصمیم می‌گیرند با ۱,۰۰۰ نفر تماس بگیرند تا چند سؤال بپرسند، آن‌ها صرفاً ۱,۰۰۰ نفر را به طور تصادفی از دفترچه تلفن انتخاب نمی‌کنند.

آن‌ها تلاش می‌کنند مطمئن شوند که این ۱,۰۰۰ نفر، با توجه به سؤالاتی که قرار است پرسیده شوند، نماینده‌ای واقعی از کل جمعیت باشند. برای مثال، جمعیت ایالات متحده تقریباً شامل ۵۱/۱% زن و ۴۸/۹% مرد است، بنابراین یک نظرسنجی معتبر سعی می‌کند این نسبت را در نمونه حفظ کند: ۵۱۱ زن و ۴۸۹ مرد (به ویژه اگر احتمال دارد پاسخ‌ها بین جنسیت‌ها متفاوت باشد). این روش نمونه‌گیری، «نمونه‌گیری طبقه‌ای (*stratified sampling*)» نامیده می‌شود؛ جمعیت به زیرگروه‌های همگن به نام «طبقه‌ها یا لایه‌ها (*strata*)» تقسیم می‌شود و تعداد مناسبی از نمونه‌ها از هر طبقه برداشته می‌شود تا اطمینان حاصل شود که مجموعه آزمون نماینده کل جمعیت است. اگر نظرسنجی‌کنندگان از نمونه‌گیری کاملاً تصادفی استفاده می‌کردند، تقریباً ۱۰/۷ % احتمال داشت که مجموعه آزمون به‌طور نامتناسب نامتقارن(*skewed*) شود، به طوری که کمتر از ۴۸/۵% یا بیش از ۵۳/۵% شرکت‌کننده زن باشد. در چنین حالتی، نتایج نظرسنجی احتمالاً جانبدارانه و غیرقابل اعتماد می‌شد.

از آنجا که درآمد متوسط یک ویژگی عددی پیوسته است، ابتدا لازم است یک ویژگی دسته‌بندی برای درآمد ایجاد کنیم. برای این کار، ابتدا هیستوگرام درآمد متوسط را با دقت بیشتری بررسی می‌کنیم (شکل ۸-۲). مشاهده می‌کنیم که بیشتر مقادیر درآمد متوسط در بازه‌ی حدود ۱/۵ تا ۶ (معادل ۱۵,۰۰۰ تا ۶۰,۰۰۰ دلار) متمرکز شده‌اند، اما برخی مقادیر درآمد بسیار بالاتر از ۶ نیز دیده می‌شوند.

از آنجا که درآمد متوسط یک ویژگی عددی پیوسته است، ابتدا لازم است یک ویژگی دسته‌بندی برای درآمد ایجاد کنیم. با نگاهی دقیق‌تر به هیستوگرام درآمد متوسط (شکل ۲-۸)، مشاهده می‌کنیم که بیشتر مقادیر در بازه‌ی حدود ۱/۵ تا ۶ (۱۵,۰۰۰ تا ۶۰,۰۰۰ دلار) متمرکز شده‌اند، اما برخی درآمدهای متوسط بسیار بالاتر از ۶ نیز وجود دارند. مهم است که هر طبقه دارای تعداد کافی نمونه باشد؛ در غیر این صورت، تخمین اهمیت آن طبقه می‌تواند جانبدارانه باشد. بنابراین، نباید تعداد طبقات زیاد باشد و هر طبقه باید به اندازه کافی بزرگ باشد. برای ایجاد این ویژگی دسته‌بندی، می‌توان از تابع pd.cut() استفاده کرد و درآمد را به پنج دسته تقسیم کرد (با برچسب‌های ۱ تا ۵). دسته ۱ شامل بازه ۰ تا ۱/۵ (کمتر از ۱۵,۰۰۰ دلار) است، دسته ۲ بازه ۱/۵ تا ۳ و به همین ترتیب ادامه دارد:

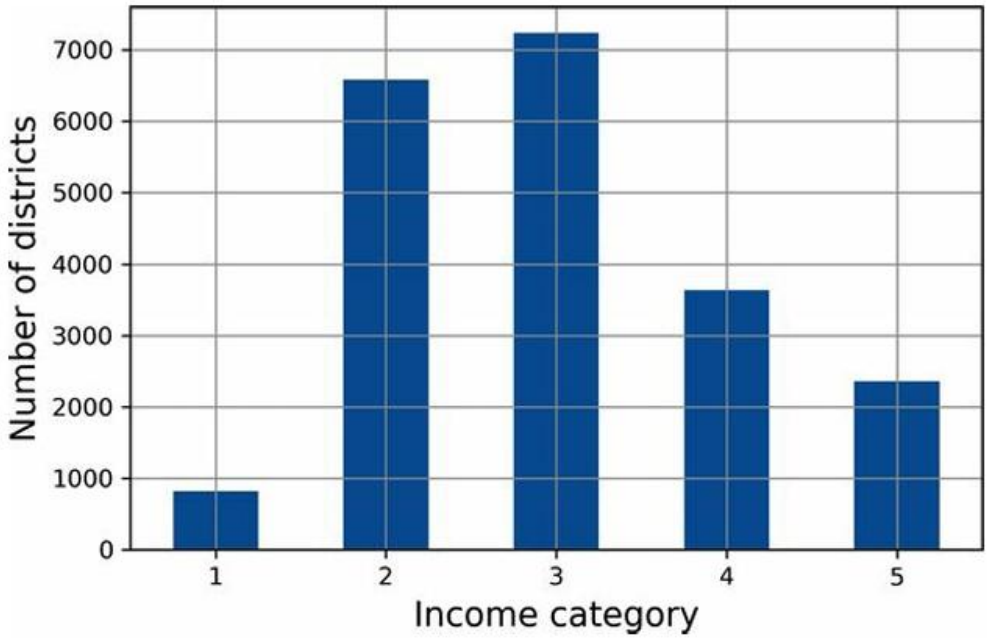
© 2025 Mohsen Zare. All rights reserved.

```
housing["income_cat"] = pd.cut(housing["median_income"],
            bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
            labels=[1, 2, 3, 4, 5])
```

این دسته‌های درآمدی در شکل ۲-۹ نمایش داده شده‌اند و نمایانگر نحوه تقسیم‌بندی نمونه‌ها بر اساس بازه‌های درآمد متوسط هستند.

```
housing["income_cat"].value_counts().sort_index().plot.bar(rot=0, grid=True)
plt.xlabel("Income category")
plt.ylabel("Number of districts")
plt.show()
```

اکنون می‌توانید نمونه‌گیری طبقه‌ای را بر اساس دسته‌بندی درآمد انجام دهید. کتابخانه Scikit-Learn چندین کلاس در ماژول sklearn.model_selection ارائه می‌دهد که روش‌های مختلف تقسیم مجموعه داده به مجموعه‌های آموزشی و آزمایشی را پیاده‌سازی می‌کنند. هر یک از این کلاس‌ها دارای متد split() هستند که تکرارکننده‌ای برمی‌گرداند و به کمک آن می‌توان تقسیم‌های مختلف آموزش و آزمونِ همان داده‌ها را ایجاد کرد.



شکل ۹-۲. هیستوگرام دسته‌های درآمدی

به طور دقیق، متد split() فقط شاخص‌های آموزش و آزمون را برمی‌گرداند و خود داده‌ها را ارائه نمی‌کند. داشتن چند تقسیم‌بندی مختلف می‌تواند مفید باشد اگر بخواهید تخمین دقیق‌تری از عملکرد

© 2025 Mohsen Zare. All rights reserved.

مدل به دست آورید، همانطور که بعداً در این فصل در مورد اعتبارسنجی متقاطع (cross-validation) توضیح داده خواهد شد. برای مثال، کد زیر ۱۰ تقسیم‌بندی طبقه‌ای متفاوت از یک مجموعه داده یکسان تولید می‌کند:

```
from sklearn.model_selection import StratifiedShuffleSplit

splitter = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
strat_splits = []
for train_index, test_index in splitter.split(housing, housing["income_cat"]):
  strat_train_set_n = housing.iloc[train_index]
  strat_test_set_n = housing.iloc[test_index]
  strat_splits.append([strat_train_set_n, strat_test_set_n])
```

در حال حاضر، می‌توانید تنها از اولین تقسیم‌بندی استفاده کنید:

```
strat_train_set, strat_test_set = strat_splits[0]
```

از آنجایی که نمونه‌گیری طبقه‌ای به‌طور گسترده استفاده می‌شود، می‌توانید به‌صورت ساده‌تر یک تقسیم واحد ایجاد کنید. برای این کار کافی است از تابع train_test_split() همراه با آرگومان stratify استفاده کنید:

```
strat_train_set, strat_test_set = train_test_split(
  housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)
```

حال می‌توانیم بررسی کنیم که آیا تقسیم‌بندی طبق انتظار انجام شده است یا خیر. برای شروع، می‌توانید نسبت نمونه‌ها در هر دسته‌بندی درآمدی را در مجموعه آزمون بررسی کنید:

```
>>> strat_test_set["income_cat"].value_counts() / len(strat_test_set)
3    0.350533
2    0.318798
4    0.176357
5    0.114341
1    0.039971
Name: income_cat, dtype: flo
```

© 2025 Mohsen Zare. All rights reserved.

با استفاده از کد مشابه، می‌توانید نسبت نمونه‌ها در هر دسته‌بندی درآمدی را برای کل مجموعه داده‌ها نیز محاسبه کنید. شکل ۲-۱۰ این نسبت‌ها را مقایسه می‌کند: نسبت‌های دسته‌بندی درآمدی در کل مجموعه داده‌ها، در مجموعه آزمون تولید شده با نمونه‌گیری طبقه‌ای، و در مجموعه آزمون حاصل از نمونه‌گیری کاملاً تصادفی. همانطور که مشاهده می‌کنید، مجموعه آزمون طبقه‌ای نسبت‌های دسته‌بندی را تقریباً مشابه کل داده‌ها حفظ می‌کند، در حالی که مجموعه آزمون کاملاً تصادفی دچار «عدم توازن» شده است.

Income Category	Overall %	Stratified %	Random %	Strat. Error %	Rand. Error %
1	3.98	4.00	4.24	0.36	6.45
2	31.88	31.88	30.74	-0.02	-3.59
3	35.06	35.05	34.52	-0.01	-1.53
4	17.63	17.64	18.41	0.03	4.42
5	11.44	11.43	12.09	-0.08	5.63

شکل ۲-۱۰ نمونه‌گیری طبقه‌ای را با نمونه‌گیری کاملاً تصادفی مقایسه می‌کند و نشان می‌دهد که نمونه‌گیری طبقه‌ای بایاس کمتری دارد و نسبت‌های دسته‌بندی درآمدی را بهتر حفظ می‌کند.

از آنجایی که دیگر به ستون income_cat نیاز ندارید، می‌توانید آن را حذف کنید و داده‌ها را به حالت اولیه خود بازگردانید:

```
for set_ in (strat_train_set, strat_test_set):
  set_.drop("income_cat", axis=1, inplace=True)
```

ما زمان زیادی را صرف تولید مجموعه آزمون کردیم، و این کار دلیل خوبی دارد: این مرحله اغلب نادیده گرفته می‌شود، اما یکی از بخش‌های حیاتی هر پروژه یادگیری ماشین است. علاوه بر این، بسیاری از مفاهیم این بخش در ادامه، هنگام بحث درباره اعتبارسنجی متقاطع، بسیار کاربردی خواهند بود. حالا وقت آن رسیده که به مرحله بعدی یعنی کاوش داده‌ها بپردازیم.

© 2025 Mohsen Zare. All rights reserved.

کاوش و تجسم داده‌ها برای کسب بینش

تاکنون فقط نگاهی سریع به داده‌ها انداخته‌اید تا درک کلی از نوع داده‌هایی که با آنها کار می‌کنید به دست آورید. حالا هدف این است که کمی عمیق‌تر بررسی کنیم.

ابتدا مطمئن شوید که مجموعه آزمون کنار گذاشته شده و شما فقط با مجموعه آموزشی کار می‌کنید. اگر مجموعه آموزشی بسیار بزرگ باشد، می‌توانید یک نمونه اکتشافی کوچک‌تر تهیه کنید تا پردازش و دستکاری داده‌ها در طول مرحله اکتشاف سریع‌تر و راحت‌تر انجام شود. در این پروژه، مجموعه آموزشی نسبتاً کوچک است، بنابراین می‌توانید مستقیماً روی کل مجموعه کار کنید. چون قصد دارید آزمایش‌های مختلفی روی تبدیل‌های کل مجموعه آموزشی انجام دهید، بهتر است یک کپی از نسخه اصلی داده‌ها تهیه کنید تا در صورت نیاز بتوانید به آن بازگردید.

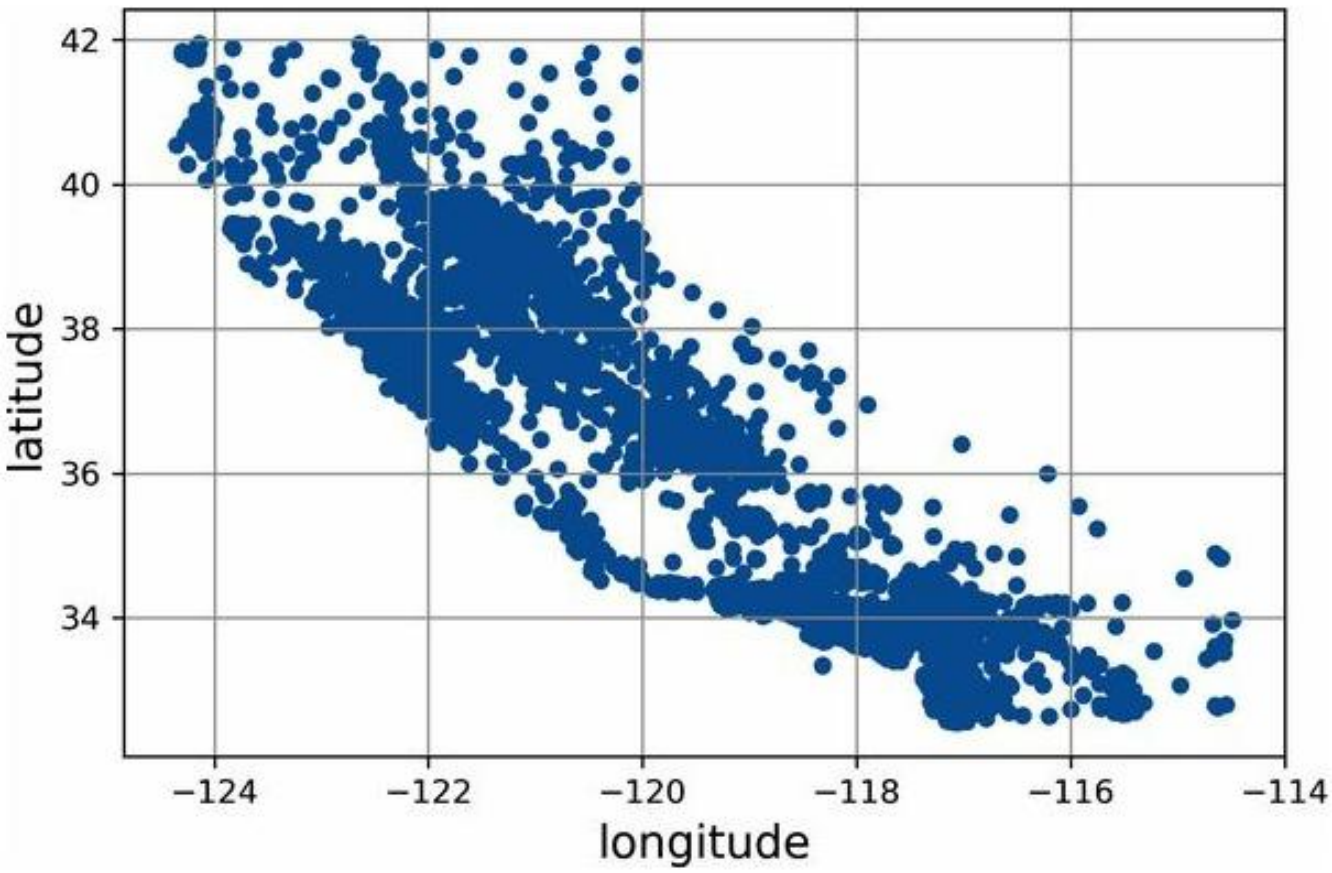
```
housing = strat_train_set.copy()
```

© 2025 Mohsen Zare. All rights reserved.

تجسم داده‌های جغرافیایی

با توجه به اینکه مجموعه داده‌ها شامل اطلاعات جغرافیایی مانند طول و عرض جغرافیایی است، مفید است که یک نمودار پراکندگی از تمام مناطق رسم کنید تا بتوانید الگوها و توزیع داده‌ها را بهتر مشاهده کنید (شکل ۲-۱۱):

```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True)
plt.show()
```



شکل ۲-۱۱ نشان‌دهنده نمودار پراکندگی جغرافیایی داده‌ها است.

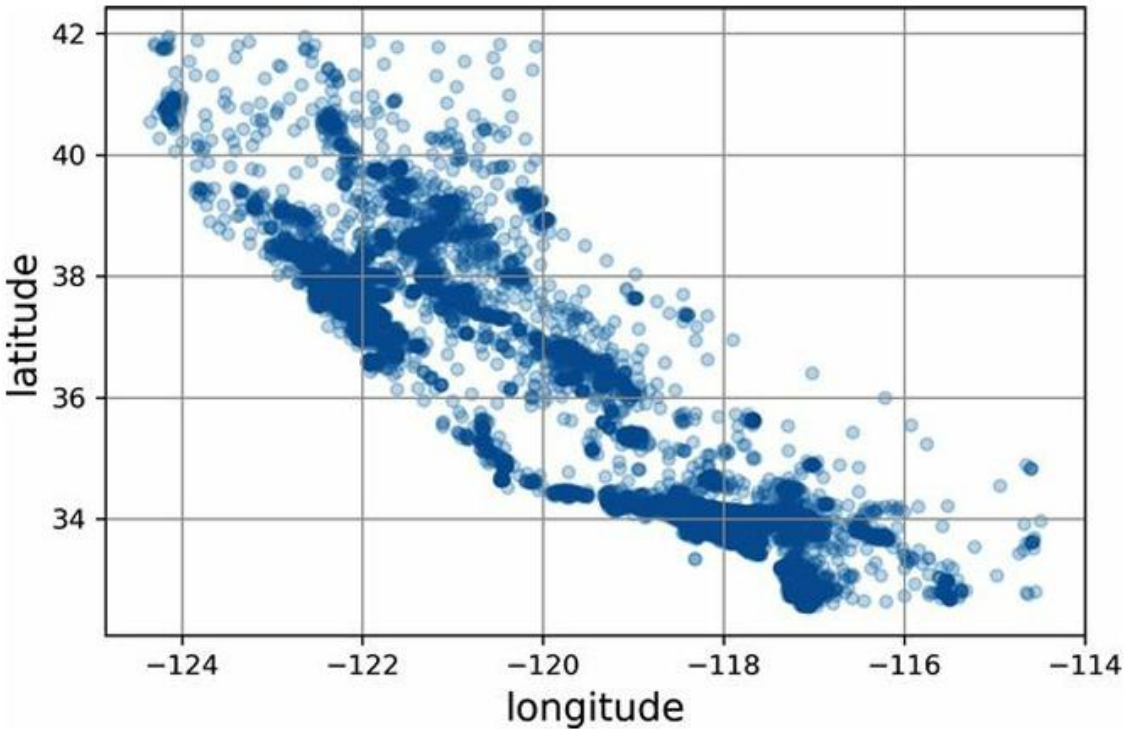
این نمودار تقریباً شکل کلی ایالت کالیفرنیا را نشان می‌دهد، اما تشخیص الگوهای خاص در آن دشوار است. با تنظیم گزینه آلفا روی ۰/۲، نقاط پر تراکم واضح‌تر می‌شوند و مشاهده مناطقی با تجمع بالای داده‌ها آسان‌تر می‌گردد (شکل ۲-۱۲):

© 2025 Mohsen Zare. All rights reserved.

```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True, alpha=0.2)
plt.show()
```

حالا تصویر واضح‌تر شده است: می‌توانید به وضوح مناطق پرجمعیت مانند خلیج سانفرانسیسکو، اطراف لس‌آنجلس و سن دیگو را مشاهده کنید، به علاوه یک نوار طولانی از مناطق با جمعیت متوسط در دره مرکزی، به ویژه حوالی ساکرامنتو و فرزنو، نیز مشخص است.

مغز انسان در شناسایی الگوها در تصاویر بسیار توانمند است، اما گاهی لازم است پارامترهای نمودار را تغییر دهید تا این الگوها واضح‌تر و برجسته‌تر شوند.



شکل ۲-۱۲. نموداری که با برجسته‌سازی تراکم نقاط، مناطق پرجمعیت را واضح‌تر نشان می‌دهد.

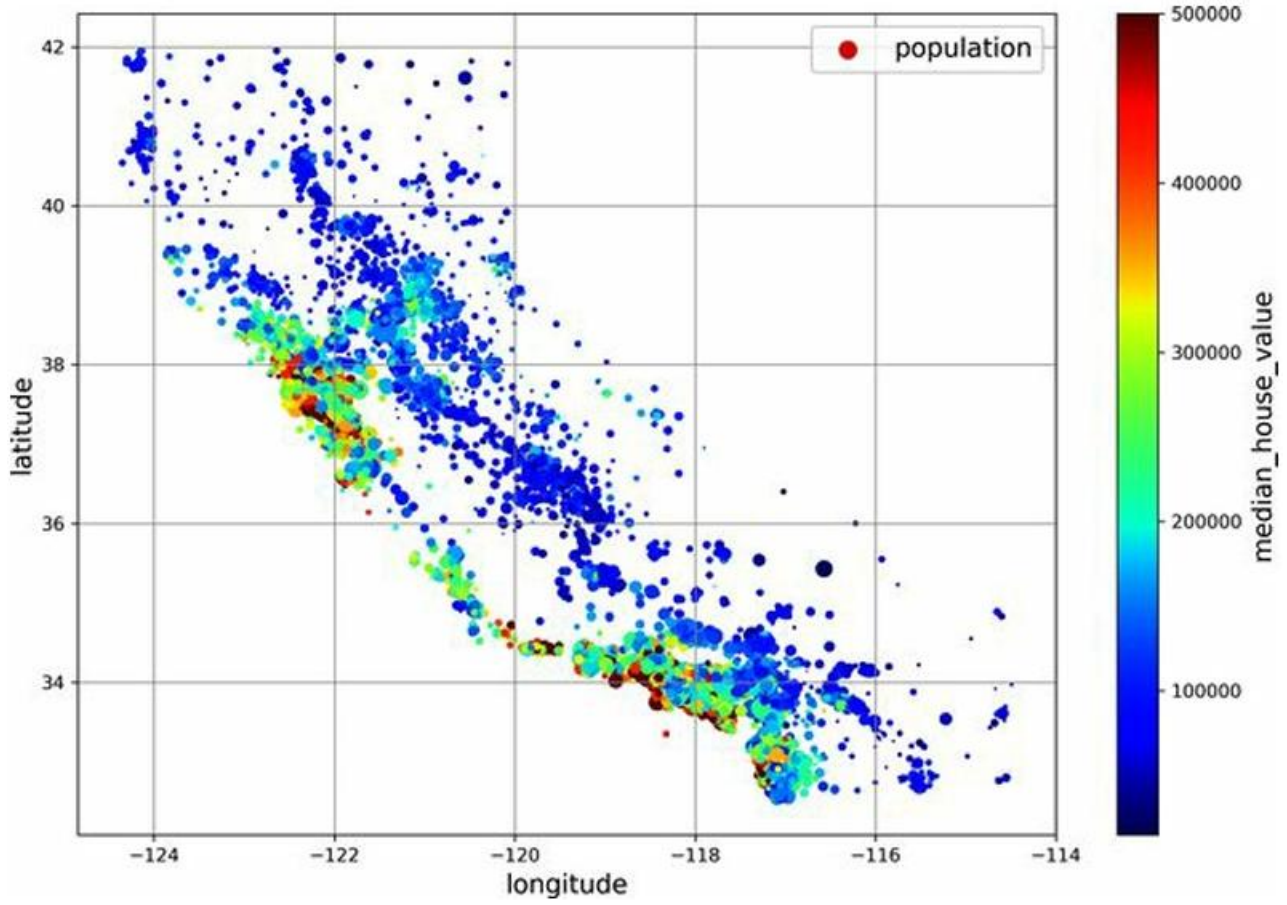
در مرحله بعد، به بررسی قیمت مسکن می‌پردازید (شکل ۲-۱۳). در این نمودار، شعاع هر دایره نشان‌دهنده جمعیت منطقه (با استفاده از گزینه s) و رنگ آن نشان‌دهنده قیمت مسکن (با استفاده از گزینه c) است. برای رنگ‌بندی از یک نقشه رنگی از پیش تعریف شده (گزینه cmap) به نام jet استفاده می‌کنید که مقادیر پایین را با رنگ آبی و مقادیر بالاتر را با رنگ قرمز نمایش می‌دهد:

```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True,
    s=housing["population"] / 100, label="population",
```

© 2025 Mohsen Zare. All rights reserved.

```
    c="median_house_value", cmap="jet", colorbar=True,
    legend=True, sharex=False, figsize=(10, 7))
plt.show()
```

این نمودار نشان می‌دهد که قیمت مسکن به شدت به موقعیت جغرافیایی (مثلاً نزدیکی به اقیانوس) و تراکم جمعیت وابسته است، همان‌طور که احتمالاً قبلاً می‌دانستید. الگوریتم‌های خوشه‌بندی می‌توانند در شناسایی خوشه‌های اصلی مفید باشند و ویژگی‌های جدیدی ایجاد کنند که فاصله یا نزدیکی هر منطقه به مرکز خوشه‌ها را اندازه‌گیری می‌کنند. همچنین، ویژگی نزدیکی به اقیانوس می‌تواند اطلاعات مفیدی بدهد، اگرچه در شمال کالیفرنیا قیمت مسکن در مناطق ساحلی همیشه بالا نیست و بنابراین نمی‌توان آن را به یک قانون ساده تعمیم داد.



شکل ۲-۱۳ نشان می‌دهد که قیمت مسکن در کالیفرنیا چگونه توزیع شده است: رنگ قرمز مربوط به خانه‌های گران، رنگ آبی مربوط به خانه‌های ارزان و اندازه دایره‌ها نشان‌دهنده جمعیت هر منطقه است؛ دایره‌های بزرگ‌تر نمایانگر مناطق پرجمعیت‌تر هستند.

© 2025 Mohsen Zare. All rights reserved.

به دنبال روابط همبستگی باشید

از آنجایی که مجموعه داده‌ها بزرگ نیست، می‌توان به راحتی «ضریب همبستگی استاندارد (*standard correlation coefficient*)»، که به آن « r پیِرسون » نیز گفته می‌شود، را بین هر جفت ویژگی با استفاده از متد corr() محاسبه کرد:

```
corr_matrix = housing.corr()
```

اکنون می‌توانید میزان ارتباط هر ویژگی با میانگین ارزش خانه را مشاهده کنید:

```
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value    1.000000
median_income         0.688380
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
longitude            -0.050859
latitude             -0.139584
Name: median_house_value, dtype: float64
```

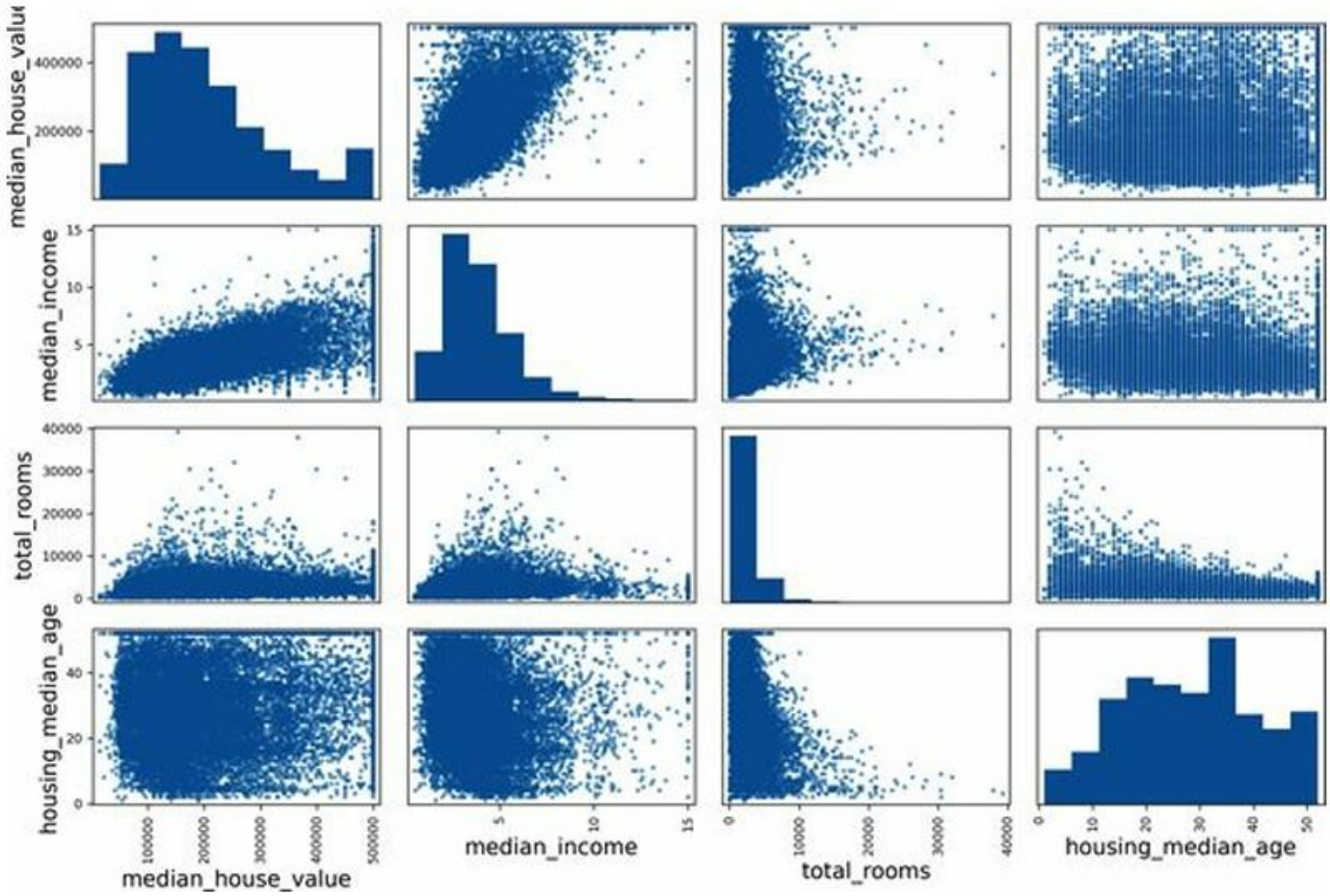
ضریب همبستگی مقداری بین ۱- و ۱ است. اگر ضریب نزدیک ۱ باشد، نشان‌دهنده همبستگی مثبتِ قوی است؛ به این معنا که با افزایش یک ویژگی، ویژگی دیگر نیز به طور قابل توجهی افزایش می‌یابد. برای مثال، با افزایش درآمد متوسط، ارزش متوسط خانه نیز افزایش می‌یابد. اگر ضریب نزدیک به ۱- باشد، نشان‌دهنده همبستگی منفیِ قوی است؛ برای مثال، می‌توان یک همبستگی منفی جزئی بین عرض جغرافیایی و ارزش متوسط خانه مشاهده کرد، یعنی با حرکت به سمت شمال، قیمت‌ها تمایل کمی به کاهش دارند. ضرایبی که نزدیک صفر هستند نشان می‌دهند که هیچ همبستگی خطی قابل توجهی بین دو ویژگی وجود ندارد.

یک روش دیگر برای بررسی همبستگی بین ویژگی‌ها، استفاده از تابع scatter_matrix() در Pandas است. این تابع برای هر جفت ویژگی عددی یک نمودار پراکندگی رسم می‌کند. با توجه به اینکه اکنون ۱۱ ویژگی عددی وجود دارد، کلِ $11^2 = 121$ نمودار ایجاد می‌شود که نمایش همه آن‌ها روی یک صفحه

© 2025 Mohsen Zare. All rights reserved.

عملی نیست. بنابراین، بهتر است روی چند ویژگی کلیدی تمرکز کنید که به نظر می‌رسد بیشترین همبستگی را با ارزش متوسط خانه دارند.

```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
          "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```



شکل ۲-۱۴. این ماتریس پراکندگی، برای هر ویژگی عددی، نمودار پراکندگی آن را در مقابل هر ویژگی عددی دیگر رسم می‌کند و روی قطر اصلی ماتریس، هیستوگرام مقادیر همان ویژگی عددی نمایش داده می‌شود.

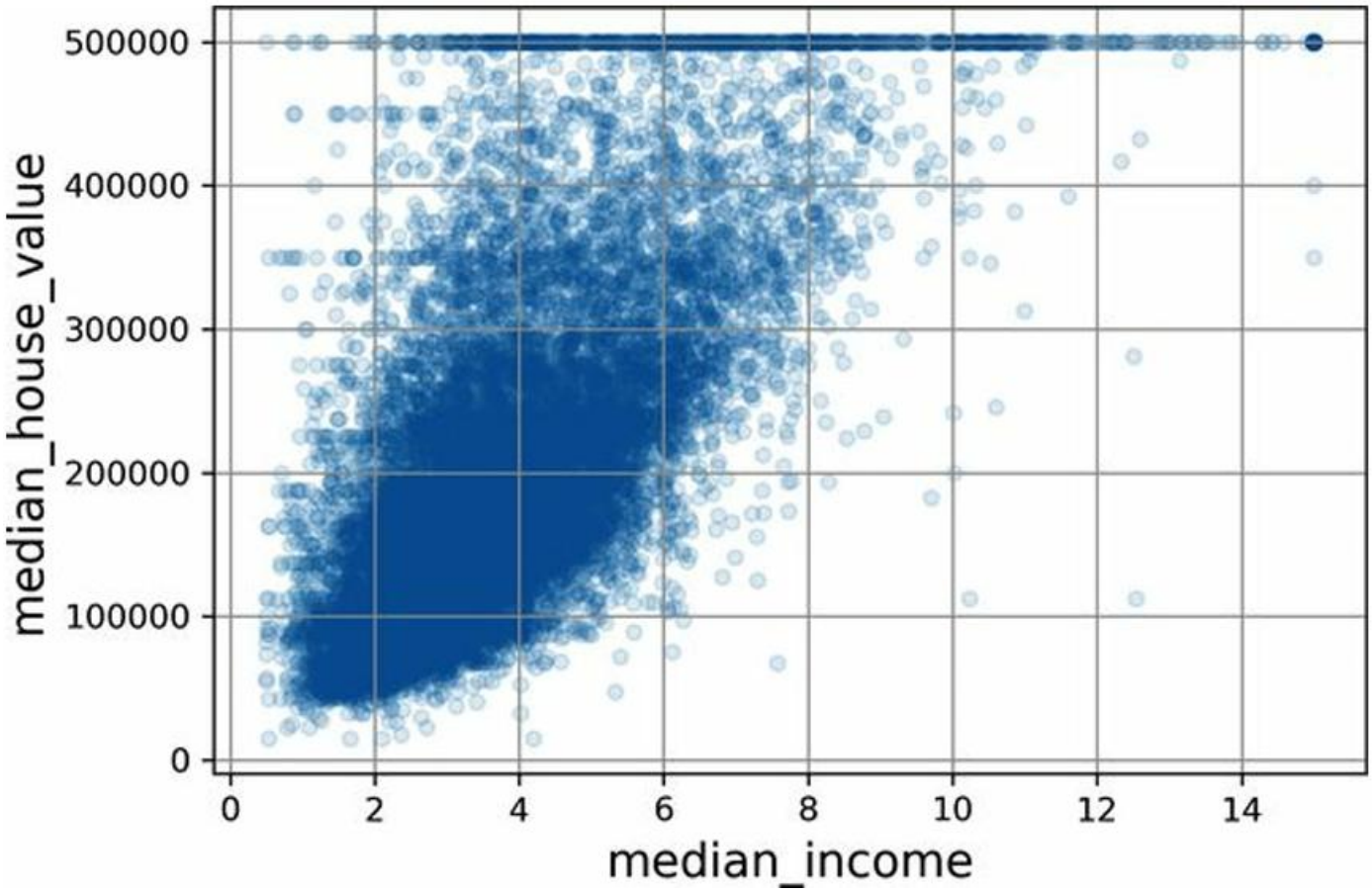
اگر Pandas هر ویژگی را در مقابل خودش رسم کند، قطر اصلی ماتریس پراکندگی پر از خطوط مستقیم خواهد بود که اطلاعات زیادی ارائه نمی‌دهد. به جای آن، Pandas روی قطر اصلی، یک

© 2025 Mohsen Zare. All rights reserved.

هیستوگرام از مقادیر هر ویژگی را نمایش می‌دهد. برای گزینه‌ها و تنظیمات بیشتر، می‌توانید به مستندات Pandas مراجعه کنید.

با بررسی نمودارهای پراکندگی همبستگی، به نظر می‌رسد امیدوارکننده‌ترین ویژگی برای پیش‌بینی ارزش متوسط خانه، درآمد متوسط باشد. بنابراین تصمیم می‌گیرید روی نمودار پراکندگی بین این دو ویژگی زوم کنید:

```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1, grid=True)
plt.show()
```



شکل ۲-۱۵ نمودار پراکندگی میانگین درآمد در مقابل میانگین ارزش خانه را نشان می‌دهد.

این بخش توضیح می‌دهد که نمودار پراکندگی میانگین درآمد و میانگین ارزش خانه چند الگوی مهم را روشن می‌کند:

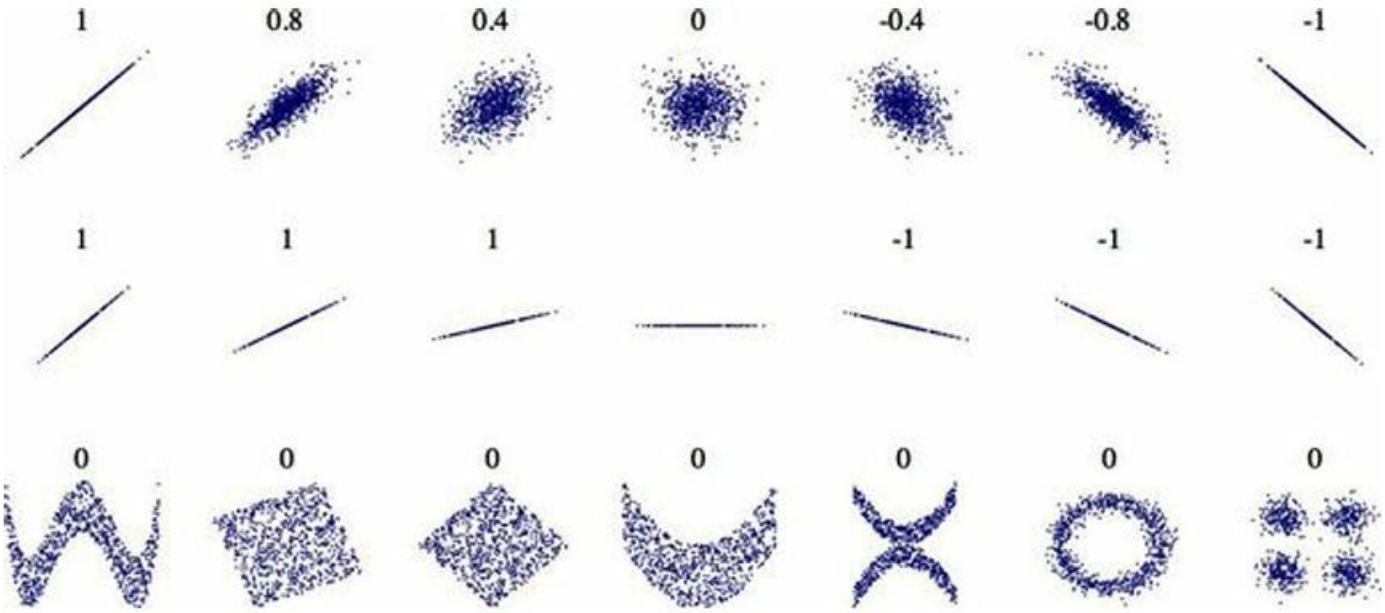
© 2025 Mohsen Zare. All rights reserved.

- همبستگی قوی: یک روند صعودی واضح و پراکندگی کم، نشان‌دهنده ارتباط مستقیم و محکم بین درآمد و قیمت مسکن.
- سقف قیمتی آشکار: یک خط افقی مشخص در حدود ۵۰۰,۰۰۰ دلار، که بیانگر محدودیت یا ماکسیمم ثبت‌شده در داده‌ها است.
- خطوط قیمتی پنهان‌تر: چند آستانه دیگر، مانند حدود ۴۵۰,۰۰۰، ۳۵۰,۰۰۰ و ۲۸۰,۰۰۰ دلار، که به‌صورت خطوط افقی ضعیف‌تر دیده می‌شوند.
- پیشنهاد پاک‌سازی داده: حذف این نقاط می‌تواند از یادگیری ناخواسته الگوریتم در بازتولید این مرزهای غیرواقعی جلوگیری کند.

هشدار

ضریب همبستگی فقط میزان رابطه‌ی خطی بین دو متغیر را اندازه‌گیری می‌کند. اگر این رابطه غیرخطی باشد، حتی در صورت وجود وابستگی قوی، مقدار همبستگی می‌تواند نزدیک به صفر باشد؛ مانند حالتی که با نزدیک شدن x به صفر، مقدار y افزایش می‌یابد، اما الگو به صورت منحنی است. در شکل ۱۶-۲، نمودارهای ردیف پایین با وجود وابستگی آشکار بین محورها، همبستگی صفر دارند، زیرا رابطه آن‌ها غیرخطی است. همچنین، همبستگی کامل با مقدار ۱ یا ۱- صرفاً نشان‌دهنده یک رابطه خطی کامل است و ارتباطی با شیب نمودار ندارد. برای نمونه، قد یک فرد بر حسب اینچ، فوت یا نانومتر، بدون توجه به مقیاس، همیشه ضریب همبستگی ۱ دارد.

© 2025 Mohsen Zare. All rights reserved.



شکل ۲-۱۶. نمونه‌هایی از ضریب همبستگی استاندارد در مجموعه‌داده‌های گوناگون (منبع: ویکی‌پدیا؛ تصویر در مالکیت عمومی)

© 2025 Mohsen Zare. All rights reserved.

آزمایش با ترکیب ویژگی‌ها

امید است بخش‌های پیشین ایده‌ای کلی از چند روش برای کاوش داده‌ها و دستیابی به بینش‌های تحلیلی ارائه کرده باشند. در این مسیر، چند ویژگی غیرعادی در داده‌ها شناسایی شد که احتمالاً لازم است پیش از ارسال به الگوریتم‌های یادگیری ماشین اصلاح شوند. همچنین، همبستگی‌های معناداری میان برخی ویژگی‌ها و به‌ویژه با ویژگی هدف مشاهده گردید. افزون بر این، مشخص شد که بعضی ویژگی‌ها دارای توزیع با عدم توازن به راست (skewed-right) هستند–نشان‌دهنده این است که اکثریت داده‌ها در سمت چپ و تعداد کمتری در سمت راست توزیع قرار دارند و دُم طولانی به سمت مقادیر بالاتر است. به بیان دیگر، معمولاً مقادیر پرتِ مثبت در سمت راست نمودار هستند. بنابراین، ممکن است نیاز باشد آن‌ها را با روش‌هایی مانند محاسبه‌ی لگاریتم یا ریشه‌ی دوم تغییر دهید. بدیهی است که میزان تلاش و دقت در این مرحله، با توجه به هر پروژه، می‌تواند متفاوت باشد، اما ایده‌های کلی معمولاً مشابه باقی می‌مانند.

آخرین گامی که ممکن است پیش از آماده‌سازی داده‌ها برای الگوریتم‌های یادگیری ماشین انجام دهید، آزمودن ترکیب‌های مختلفی از ویژگی‌ها است. برای نمونه، اگر تعداد خانوارهای موجود در یک منطقه مشخص نباشد، آگاهی از تعداد کل اتاق‌ها چندان سودمند نخواهد بود. آنچه واقعاً اهمیت دارد، «تعداد اتاق‌ها به ازای هر خانوار» است. به همین شکل، تعداد کل اتاق‌خواب‌ها به تنهایی اطلاعات کافی ارائه نمی‌دهد و بهتر است در نسبت با تعداد کل اتاق‌ها بررسی شود. همچنین، «جمعیت به إزای هر خانوار» نیز می‌تواند ترکیب ویژگی جالبی برای تحلیل باشد. این ویژگی‌های جدید را می‌توان به شکل زیر ایجاد کرد:

```
housing["rooms_per_house"] = housing["total_rooms"] / housing["households"]
housing["bedrooms_ratio"] = housing["total_bedrooms"] / housing["total_rooms"]
housing["people_per_house"] = housing["population"] / housing["households"]
```

و سپس بار دیگر ماتریس همبستگی را بررسی می‌کنید:

```
>>> corr_matrix = housing.corr()
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value    1.000000
median_income         0.688380
rooms_per_house       0.143663
```

© 2025 Mohsen Zare. All rights reserved.

```
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
people_per_house     -0.038224
longitude            -0.050859
latitude             -0.139584
bedrooms_ratio       -0.256397
Name: median_house_value, dtype: float64
```

نتیجه‌ی بدی نیست! ویژگی جدید bedrooms_ratio همبستگی بالاتری با میانگین ارزش خانه نسبت به تعداد کل اتاق‌ها یا تعداد اتاق‌خواب‌ها نشان می‌دهد. به نظر می‌رسد خانه‌هایی که نسبت اتاق‌خواب به کل اتاق‌هایشان کمتر است، قیمت بالاتری دارند. همچنین، تعداد اتاق‌ها به ازای هر خانوار اطلاعات دقیق‌تری نسبت به تعداد کل اتاق‌ها در یک منطقه ارائه می‌دهد؛ به طور طبیعی، هرچه خانه‌ها بزرگ‌تر باشند، قیمت آن‌ها نیز بیشتر است.

این دور از کاوش داده‌ها لازم نیست کاملاً بی‌نقص باشد؛ مهم این است که با گام‌های درست شروع کنید و سریع به بینش‌هایی برسید که به ساخت اولین نمونه اولیه‌ی قابل قبول کمک کند. این فرایند تکرارشونده است: وقتی نمونه اولیه را راه‌اندازی و اجرا کردید، خروجی آن را تحلیل کنید، بینش‌های تازه به دست آورید و دوباره به مرحله کاوش برگردید.

© 2025 Mohsen Zare. All rights reserved.

آماده‌سازی داده‌ها برای الگوریتم‌های یادگیری ماشین

وقت آن رسیده است که داده‌ها را برای الگوریتم‌های یادگیری ماشین آماده کنید. به جای انجام این کار به صورت دستی، بهتر است توابعی برای این منظور بنویسید، و دلایل خوبی برای این کار وجود دارد:

- این کار به شما اجازه می‌دهد همان تبدیل‌ها را به راحتی روی هر مجموعه داده‌ای اعمال کنید، مثلاً وقتی دفعه بعد یک مجموعه داده جدید دریافت می‌کنید.
- به تدریج می‌توانید یک کتابخانه از توابع تبدیل بسازید که در پروژه‌های آینده دوباره قابل استفاده باشد.
- می‌توانید از این توابع در سیستم‌های زنده برای آماده‌سازی داده‌های جدید پیش از ارسال به الگوریتم‌ها استفاده کنید.
- این روش به شما امکان می‌دهد به راحتی ترکیب‌های مختلف تبدیل‌ها را امتحان کرده و ببینید کدام ترکیب بهترین نتیجه را ارائه می‌دهد.

اما ابتدا، باید به یک نسخه‌ی تمیز از مجموعه‌ی آموزشی برگردید، برای مثال با کپی دوباره strat_train_set. همچنین لازم است پیش‌بینی‌کننده‌ها و برچسب‌ها را از هم جدا کنید، زیرا ممکن است نخواهید همان تبدیل‌ها را روی پیش‌بین‌ها و مقادیر هدف اعمال کنید. توجه داشته باشید که متد ()drop یک کپی از داده‌ها ایجاد می‌کند و روی strat_train_set اصلی تأثیری نمی‌گذارد:

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

© 2025 Mohsen Zare. All rights reserved.

پاکسازی داده‌ها

اکثر الگوریتم‌های یادگیری ماشین نمی‌توانند با «ویژگی‌های مفقود (missing features)» یا «مقادیرِ ناموجود» کار کنند، بنابراین باید این موارد را مدیریت کنید. برای مثال، قبلاً مشاهده کردید که ویژگی total_bedrooms شامل مقادیری از دست رفته است. برای رفع این مشکل سه راه وجود دارد:

۱. حذف مناطق (district) مربوطه.

۲. حذف کامل ویژگی مورد نظر.

۳. پر کردن مقادیر مفقود با یک مقدار مناسب (مثل صفر (zero)، میانگین (mean)، میانه (median) و غیره) که به آن روش «تخمینِ داده‌های مفقود (*imputation*)»—فرآیند جایگزین کردن مقادیر ناموجود و داده‌های ناقص با مقادیر تخمینی یا محاسبه‌شده—گفته می‌شود.

شما می‌توانید این موارد را به راحتی با استفاده از متدهای dropna()، drop() و fillna() در قاب داده Pandas (Pandas DataFrame) انجام دهید:

```
housing.dropna(subset=["total_bedrooms"], inplace=True) # option 1

housing.drop("total_bedrooms", axis=1) # option 2

median = housing["total_bedrooms"].median() # option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

شما تصمیم می‌گیرید گزینه ۳ را انتخاب کنید زیرا کمترین میزان تخریب را دارد، اما به جای روش‌های سنتی، از یک کلاس مفید در Scikit-Learn به نام SimpleImputer استفاده می‌کنید. مزیت این روش این است که مقدار میانه هر ویژگی را ذخیره می‌کند و این امکان را فراهم می‌کند که مقادیر مفقود را نه تنها در مجموعه آموزشی، بلکه در مجموعه اعتبارسنجی، مجموعه آزمون و هر داده جدیدی که به مدل داده می‌شود، جایگزین کنید. برای استفاده از این کلاس، ابتدا یک نمونه از SimpleImputer ایجاد می‌کنید و مشخص می‌کنید که می‌خواهید مقادیر مفقود هر ویژگی با میانه همان ویژگی جایگزین شود:

© 2025 Mohsen Zare. All rights reserved.

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
```

از آنجایی که میانه فقط برای ویژگی‌های عددی قابل محاسبه است، ابتدا یک کپی از داده‌ها ایجاد می‌کنید که تنها شامل ویژگی‌های عددی باشد. این کار به طور خودکار ویژگی متنی ocean_proximity را حذف می‌کند:

```
housing_num = housing.select_dtypes(include=[np.number])
```

اکنون می‌توانید نمونه‌ی SimpleImputer را با استفاده از متد fit() روی داده‌های آموزشی برازش دهید:

```
imputer.fit(housing_num)
```

«تخمین‌گر داده‌های مفقود (imputer)»—الگوریتم یا ابزاری که مقادیر از دست‌رفته را تخمین می‌زند و جایگزین می‌کند؛ به زبان ساده‌تر داده های ناموجود را پُر می‌کند—به سادگی میانه‌ی هر ویژگی را محاسبه کرده و آن را در متغیر داخلی statistics_ instance ذخیره می‌کند. هرچند در حال حاضر تنها ویژگی total_bedrooms مقادیر مفقود دارد، اما نمی‌توان مطمئن بود که در داده‌های جدید هیچ مقدارِ ناموجودی وجود نداشته باشد. بنابراین، اعمال تخمین‌گر داده‌های مفقود به تمام ویژگی‌های عددی امن‌تر است:

```
>>> imputer.statistics_
array([-118.51 , 34.26 , 29. , 2125. , 434. , 1167. , 408. , 3.5385])
>>> housing_num.median().values
array([-118.51 , 34.26 , 29. , 2125. , 434. , 1167. , 408. , 3.5385])
```

اکنون می‌توانید از این تخمین‌گر داده‌های مفقود که قبلاً «آموزش‌دیده» است، برای تبدیل مجموعه‌ی آموزشی استفاده کنید؛ در این تبدیل، مقادیر مفقود با میانه‌های محاسبه‌شده جایگزین می‌شوند:

```
X = imputer.transform(housing_num)
```

مقادیر مفقود می‌توانند با استفاده از راهبرد(strategy) های مختلف جایگزین شوند:

© 2025 Mohsen Zare. All rights reserved.

- استفاده از میانگین هر ویژگی (strategy = "mean")
- استفاده از پرتکرارترین مقدار ویژگی (strategy = "most_frequent")
- یا جایگزینی با یک مقدار ثابت مشخص (strategy = "constant", fill_value = ...)

دو استراتژی آخر قابلیت پشتیبانی از داده‌های غیرعددی را نیز دارند.

نکته

همچنین imputer های پیشرفته‌تری در پکیج sklearn.impute موجود هستند که فقط روی ویژگی‌های عددی کار می‌کنند: ۱- KNNImputer : هر مقدار مفقودی را با میانگین مقادیر k-نزدیک‌ترین همسایه برای همان ویژگی جایگزین می‌کند. فاصله بین نمونه‌ها براساس تمام ویژگی‌های موجود محاسبه می‌شود. ۲- IterativeImputer : برای هر ویژگی، یک مدل رگرسیون آموزش می‌دهد تا مقادیر ناموجود را بر اساس سایر ویژگی‌ها پیش‌بینی کند. سپس مدل را روی داده‌های به‌روز شده دوباره آموزش می‌دهد و این فرآیند را چندین بار تکرار می‌کند تا مدل‌ها و مقادیرِ جایگزین به تدریج بهبود یابند.

طراحی SCIKIT-LEARN

رابط برنامه‌نویسی(API) Scikit-Learn به‌طور قابل توجهی خوب طراحی شده است. قواعد اصلی طراحی آن به شرح زیر است:

یکپارچگی و سازگاری (*Consistency*)

تمام اشیاء یک رابط ساده و یکنواخت دارند:

تخمین‌گرها (*Estimators*)

© 2025 Mohsen Zare. All rights reserved.

هر شیئی که بتواند برخی پارامترها را بر اساس یک مجموعه داده تخمین بزند، یک «تخمین‌گر (*estimator*)» نامیده می‌شود. برای مثال، SimpleImputer یک تخمین‌گر است. فرآیند تخمین با استفاده از متد ()fit انجام می‌شود و یک مجموعه داده را به عنوان پارامتر دریافت می‌کند؛ در الگوریتم‌های یادگیری نظارت‌شده، ممکن است مجموعه داده دوم که شامل برچسب‌ها است نیز مورد استفاده قرار گیرد. هر پارامتر دیگری که برای هدایت فرآیند تخمین لازم است، یک هایپرپارامتر محسوب می‌شود، مانند استراتژی SimpleImputer، و معمولاً از طریق پارامتر سازنده(constructor) شیء تنظیم می‌شود.

ترنسفورمرها (*Transformers*)

برخی از تخمین‌گرها، مانند SimpleImputer، علاوه بر تخمین پارامترها، قادر به اعمال تبدیل روی مجموعه داده‌ها—یعنی تغییر شکل یا بازنمایی داده‌ها بدون تغییر ماهیت اصلی آن‌ها—نیز هستند؛ به این دسته‌ها «ترنسفورمرها(*transformers*)» گفته می‌شود. تبدیل با استفاده از متد ()transform انجام می‌شود و مجموعه داده‌ای که باید تبدیل شود به عنوان پارامتر به آن داده می‌شود. این متد، مجموعه داده تبدیل‌شده را برمی‌گرداند و معمولاً بر اساس پارامترهای آموخته شده عمل می‌کند، همانطور که در SimpleImputer مشاهده شد. همه ترنسفورمرها همچنین متد مفیدی به نام ()fit_transform دارند که معادل فراخوانی متوالی ()fit و سپس ()transform است، و در بسیاری از موارد بهینه شده و سریع‌تر اجرا می‌شود.

پیش‌بین‌ها (*Predictors*)

برخی از تخمین‌گرها، علاوه بر قابلیت تخمین پارامترها، توانایی انجام پیش‌بینی را نیز دارند؛ به این‌ها «پیش‌بین‌ها (*predictors*)» گفته می‌شود. برای مثال، مدل رگرسیون خطی در فصل قبل یک پیش‌بین بود که با توجه به سرانه تولید ناخالص داخلی یک کشور، رضایت از زندگی را پیش‌بینی می‌کرد. یک پیش‌بین، دارای متد ()predict است که مجموعه‌ای از نمونه‌های جدید را دریافت و پیش‌بینی‌های متناظر را برمی‌گرداند. همچنین،

© 2025 Mohsen Zare. All rights reserved.

متد ()score کیفیت پیش‌بینی‌ها را با توجه به یک مجموعه آزمون و برچسب‌های مربوطه اندازه‌گیری می‌کند، که برای الگوریتم‌های یادگیری نظارت‌شده کاربرد دارد.

بازرسی(*Inspection*)

تمام هایپرپارامترهای یک تخمین‌گر به‌طور مستقیم از طریق متغیرهای نمونه عمومی قابل دسترسی هستند، برای مثال imputer.strategy . همچنین، تمام پارامترهای آموخته‌شده توسط تخمین‌گر از طریق متغیرهای نمونه عمومی با پسوند زیرخط (_) در دسترس هستند، مانند imputer.statistics_.

عدم تکثیر کلاس‌ها(*Nonproliferation of classes*)

در Scikit-Learn، داده‌ها به جای استفاده از کلاس‌های اختصاصی، معمولاً به صورت آرایه‌های NumPy یا ماتریس‌های تُنُک SciPy (scipy.sparse)—ماتریس‌های پراکنده یا خلوت با تعداد زیاد صفر—نمایش داده می‌شوند. هایپرپارامترها نیز صرفاً به شکل رشته‌ها یا اعداد معمولی پایتون نگهداری می‌شوند.

ترکیب‌بندی(*Composition*)

در Scikit-Learn، بلوک‌های سازنده (building blocks) طوری طراحی شده‌اند که حداکثر قابلیت استفاده مجدد داشته باشند. برای مثال، می‌توان به سادگی یک دنباله از ترنسفورمرها را به هم متصل کرد و سپس یک تخمین‌گر نهایی به آن اضافه کرد تا یک Pipeline ایجاد شود.

پیش‌فرض‌های معقول(*Sensible defaults*)

Scikit-Learn برای اکثر پارامترها مقادیر پیش‌فرض معقولی ارائه می‌دهد که به شما امکان می‌دهد به سرعت یک سیستم پایه و قابل استفاده ایجاد کنید.

© 2025 Mohsen Zare. All rights reserved.

ترانسفورمرهای Scikit-Learn حتی زمانی که ورودی از نوع Pandas DataFrame باشد، خروجی را به شکل آرایه NumPy (و گاهی ماتریس تُنُک SciPy) ارائه می‌دهند. بنابراین، خروجی imputer.transform(housing_num) یک آرایه NumPy است که X نه نام ستون دارد و نه اندیس. خوشبختانه، می‌توان با قرار دادن این آرایه X در یک DataFrame و استفاده از نام ستون‌ها و اندیس‌های housing_num، این اطلاعات را recall کرد:

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

© 2025 Mohsen Zare. All rights reserved.

مدیریت ویژگی‌های متنی و دسته‌ای

تاکنون تنها با ویژگی‌های عددی(numerical attributes) کار کرده‌ایم، اما داده‌های شما ممکن است شامل ویژگی‌های متنی نیز باشد. در این مجموعه داده، تنها ویژگی از این نوع ocean_proximity است. بیایید مقادیر آن را برای چند نمونه اول بررسی کنیم:

```
>>> housing_cat = housing[["ocean_proximity"]]
>>> housing_cat.head(8)
      ocean_proximity
13096       NEAR BAY
14973      <1H OCEAN
3785          INLAND
14689         INLAND
20507     NEAR OCEAN
1286          INLAND
18078      <1H OCEAN
4396        NEAR BAY
```

این ویژگی دلخواه نیست: تنها تعداد محدودی مقدار ممکن دارد که هر کدام یک دسته را نشان می‌دهند. بنابراین، این ویژگی یک ویژگی دسته‌ای محسوب می‌شود. اکثر الگوریتم‌های یادگیری ماشین ترجیح می‌دهند با مقادیر عددی کار کنند، پس بیایید این دسته‌ها را از متن به عدد تبدیل کنیم. برای این منظور می‌توانیم از کلاس OrdinalEncoder در Scikit-Learn استفاده کنیم:

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
```

چند نمونه اول از مقادیر کدگذاری‌شده در housing_cat_encoded به این صورت نمایش داده می‌شوند:

```
>>> housing_cat_encoded[:8]
```

© 2025 Mohsen Zare. All rights reserved.

```
array([[3.],
       [0.],
       [1.],
       [1.],
       [4.],
       [1.],
       [0.],
       [3.]])
```

می‌توانید لیست دسته‌ها را با استفاده از متغیر نمونه _categories مشاهده کنید. این متغیر شامل یک آرایه‌ی یک‌بُعدی از دسته‌ها برای هر ویژگی دسته‌ای است. در این مثال، چون تنها یک ویژگی دسته‌ای (ocean_proximity) داریم، _categories یک لیست شامل یک آرایه واحد از مقادیر دسته‌ای خواهد بود:

```
>>> ordinal_encoder.categories_
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
 dtype=object)]
```

یکی از مشکلات کدگذاری ترتیبی این است که الگوریتم‌های یادگیری ماشین فرض می‌کنند مقادیر نزدیک به هم، شباهت بیشتری نسبت به مقادیر دور دارند. این می‌تواند برای دسته‌های مرتب‌شده مانند «بد»، «متوسط»، «خوب» و «عالی» مناسب باشد، اما برای ستون ocean_proximity صحیح نیست؛ به عنوان مثال، دسته‌های ۰ و ۴ شباهت بیشتری به هم ندارند به نسبت دسته‌های ۰ و ۱. برای حل این مشکل، معمولاً برای هر دسته یک ویژگی باینری ایجاد می‌شود: به عنوان مثال، یک ویژگی برابر ۱ اگر دسته "<1H OCEAN" باشد و ۰ در غیر این صورت، ویژگی دیگر ۱ اگر دسته "INLAND" باشد و ۰ در غیر این صورت، و به همین ترتیب. این روش را «کدگذاری وان-هات (*one-hot encoding*)» می‌نامند—کُدگذاری وان-هات یعنی تبدیل دسته‌ها (categories) به بُردارهای باینری که فقط یک عنصر در آن ۱ است و بقیه صفر؛ مثال: دسته‌ها = {قرمز، آبی، سبز} : وان-هات برای "آبی" ⟵ [۰, ۱, ۰]— ،زیرا فقط یک ویژگی «داغ» (۱) است و بقیه «سرد» (۰). ویژگی‌های حاصل گاهی «ویژگی‌های ساختگی (*dummy attributes*)» نامیده می‌شوند. Scikit-Learn کلاس OneHotEncoder را برای تبدیل مقادیر دسته‌ای به بردارهای وان-هات ارائه می‌دهد:

```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
```

© 2025 Mohsen Zare. All rights reserved.

```
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

به طور پیش‌فرض، خروجی OneHotEncoder به جای یک آرایه NumPy، یک «ماتریس تُنُک SciPy (*SciPy sparse matrix*)»—که به آن «ماتریس پراکنده» یا «ماتریس خلوت» و یا «ماتریس کم‌چگال» هم گفته می‌شود—تولید می‌کند:

```
>>> housing_cat_1hot
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
with 16512 stored elements in Compressed Sparse Row format>
```

یک ماتریس تُنُک، روش بسیار کارآمدی برای نمایش ماتریس‌هایی است که بیشتر عناصرشان صفر هستند. در واقع، فقط مقادیر غیرصفر و موقعیت آن‌ها ذخیره می‌شود. وقتی یک ویژگی دسته‌ای صدها یا هزاران دسته دارد، کدگذاری وان-هات یک ماتریس بسیار بزرگ تولید می‌کند که در آن تقریباً همه مقادیر صفر و تنها یک مقدار ۱ در هر ردیف وجود دارد. در چنین حالتی، استفاده از ماتریس تُنُک مزایای زیادی دارد: مصرف حافظه را کاهش می‌دهد و محاسبات سریع‌تر انجام می‌شود. این ماتریس را می‌توان تقریباً مانند یک آرایه دوبُعدی معمولی استفاده کرد، و اگر نیاز داشته باشید آن را به آرایه NumPy متراکم تبدیل کنید، کافی است متد toarray() را فراخوانی کنید:

```
>>> housing_cat_1hot.toarray()
array([[0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.]])
```

به‌طور جایگزین، می‌توانید هنگام ایجاد OneHotEncoder، پارامتر sparse=False را تنظیم کنید. در این حالت، متد transform() مستقیماً یک آرایه NumPy متراکم معمولی برمی‌گرداند.

مشابه OrdinalEncoder، می‌توانید لیست دسته‌ها را با استفاده از متغیر نمونه categories_ در OneHotEncoder دریافت کنید:

© 2025 Mohsen Zare. All rights reserved.

```
>>> cat_encoder.categories_
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

کتابخانه‌ی پانداس تابعی به نام get_dummies() ارائه می‌دهد که هر ویژگی دسته‌بندی‌شده را به یک نمایش one-hot تبدیل می‌کند؛ در این حالت، برای هر دسته یک ویژگی باینری ایجاد می‌شود:

```
>>> df_test = pd.DataFrame({"ocean_proximity": ["INLAND", "NEAR BAY"]})
>>> pd.get_dummies(df_test)
   ocean_proximity_INLAND  ocean_proximity_NEAR BAY
0                       1                         0
1                       0                         1
```

ظاهر get_dummies() ساده و کاربرپسند است، اما مزیت OneHotEncoder این است که دسته‌های آموزش دیده را به خاطر می‌سپارد. این نکته بسیار مهم است، زیرا وقتی مدل شما در محیط عملیاتی قرار می‌گیرد، باید دقیقاً همان ویژگی‌ها را دریافت کند که در طول آموزش استفاده شده‌اند، نه بیشتر و نه کمتر. برای مثال، اگر cat_encoder آموزش دیده را روی df_test با متد transform() اعمال کنیم (نه fit_transform())، دقیقاً همان ساختار ویژگی‌ها را حفظ می‌کند:

```
>>> cat_encoder.transform(df_test)
array([[0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

تفاوت واضح است: get_dummies() تنها دو دسته را در داده‌ها دیده و بنابراین فقط دو ستون ایجاد کرده است، در حالی که OneHotEncoder برای هر دسته‌ای که در مرحله آموزش یاد گرفته شده، یک ستون مجزا و با ترتیب صحیح تولید می‌کند. علاوه بر این، اگر get_dummies() را روی یک DataFrame با یک دسته ناشناخته (مثلاً "<2H OCEAN") اعمال کنید، بدون خطا یک ستون جدید برای آن ایجاد می‌کند، که می‌تواند باعث ناسازگاری با مدل آموزش‌دیده شود:

```
>>> df_test_unknown = pd.DataFrame({"ocean_proximity": ["<2H OCEAN", "ISLAND"]})
>>> pd.get_dummies(df_test_unknown)
 ocean_proximity_<2H OCEAN  ocean_proximity_ISLAND
0                          1                        0
1                          0                        1
```

© 2025 Mohsen Zare. All rights reserved.

اما OneHotEncoder هوشمندانه‌تر عمل می‌کند: وقتی با یک دسته ناشناخته مواجه می‌شود، به جای ایجاد ستون جدید، یک استثنا (Exception) تولید می‌کند. اگر بخواهید، می‌توانید با تنظیم هایپرپارامتر "handle_unknown="ignore، این رفتار را تغییر دهید؛ در این حالت، دسته ناشناخته به سادگی با صفر نمایش داده می‌شود و مشکلی در خروجی ایجاد نمی‌کند:

```
>>> cat_encoder.handle_unknown = "ignore"
>>> cat_encoder.transform(df_test_unknown)
array([[0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0.]])
```

نکته

اگر یک ویژگی دسته‌بندی‌شده تعداد زیادی دسته ممکن داشته باشد (مثلاً کد کشور، حرفه یا گونه)، کدگذاری وان-هات می‌تواند منجر به تولید تعداد زیادی ویژگی ورودی شود. این موضوع ممکن است آموزش مدل را کند کرده و حتی عملکرد آن را کاهش دهد. در چنین شرایطی، می‌توانید ویژگی دسته‌بندی‌شده را با یک ویژگی عددیِ مفیدِ مرتبط جایگزین کنید: برای مثال، به جای ویژگی ocean_proximity می‌توان فاصله تا اقیانوس را استفاده کرد، یا به جای کد کشور، جمعیت کشور و سرانه تولید ناخالص داخلی (GDP per capita) را به کار برد. به عنوان جایگزین دیگر، می‌توانید از رمزگذارهای ارائه شده توسط بسته category_encoders در GitHub استفاده کنید. همچنین، هنگام کار با شبکه‌های عصبی، هر دسته می‌تواند با یک بردار کم‌بعد و قابل یادگیری به نام « نمایش نهفته یا بردار نهفته (*embedding*)»—یعنی: نگاشت یا بازنماییِ داده‌ها گسسته (کلمات، تصاویر، کاربران، آیتم‌ها و ...) به‌صورت بردارهای پیوسته در فضای بُعد پایین‌تر، به‌گونه‌ای که شباهت‌ها و روابط معنایی در این فضا حفظ شوند—جایگزین شود؛ این نمونه‌ای از «یادگیری بازنمایی (representation learning)» است—یعنی: مدلی خودش یاد بگیرد بهترین ویژگی‌ها یا نمایش داده‌ها را استخراج کند، بدون اینکه ما به صورت دستی ویژگی‌ها (features) را تعریف کنیم—؛ برای جزئیات بیشتر می‌توانید به فصل‌های ۱۳ و ۱۷ در جلد دوم ترجمه کتاب مراجعه کنید.

زمانی که هر تخمین‌گر Scikit-Learn را با یک DataFrame آموزش می‌دهید، نام ستون‌ها در feature_names_in_ attribute ذخیره می‌شوند. این ویژگی تضمین می‌کند که هر DataFrame که بعداً به تخمین‌گر داده می‌شود—مثلاً برای transform() یا predict()—نام ستون‌های یکسانی

© 2025 Mohsen Zare. All rights reserved.

داشته باشد. همچنین، ترنسفورمرها متدی به نام get_feature_names_out() ارائه می‌دهند که می‌توان از آن برای ساخت یک DataFrame با ستون‌های خروجی ترنسفورمر استفاده کرد:

```
>>> cat_encoder.feature_names_in_
array(['ocean_proximity'], dtype=object)
>>> cat_encoder.get_feature_names_out()
array(['ocean_proximity_<1H OCEAN', 'ocean_proximity_INLAND',
 'ocean_proximity_ISLAND', 'ocean_proximity_NEAR BAY',
 'ocean_proximity_NEAR OCEAN'], dtype=object)
>>> df_output = pd.DataFrame(cat_encoder.transform(df_test_unknown),
...                          columns=cat_encoder.get_feature_names_out(),
...                          index=df_test_unknown.index)
...
```

© 2025 Mohsen Zare. All rights reserved.

مقیاس‌بندی ویژگی‌ها و تبدیل داده‌ها

یکی از مهم‌ترین تبدیل‌هایی که باید روی داده‌ها اعمال شود، «مقیاس‌بندی ویژگی‌ها (*feature scaling*)»—یعنی تغییر مقیاس داده‌ها برای هم‌تراز شدن ویژگی‌ها—است. به جز چند استثنا، اکثر الگوریتم‌های یادگیری ماشین زمانی که ویژگی‌های عددی ورودی در مقیاس‌های بسیار متفاوت باشند، عملکرد مناسبی ندارند. این مسئله در داده‌های مسکن نیز مشاهده می‌شود: تعداد کل اتاق‌ها از حدود ۶ تا ۳۹,۳۲۰ متغیر است، در حالی که میانگین درآمد تنها بین ۰ تا ۱۵ قرار دارد. بدون «مقیاس‌بندی (*Scaling*)»،—یعنی تغییر دادن بازه یا توزیع داده‌ها برای اینکه الگوریتم‌ها سریع‌تر و پایدارتر کار کنند—بیشتر مدل‌ها به سمت نادیده گرفتن میانگین درآمد و تمرکز بیش از حد روی تعداد اتاق‌ها گرایش پیدا می‌کنند.

دو روش رایج برای یکسان کردن مقیاس همه ویژگی‌ها وجود دارد: « مقیاس‌بندی مینیمم–ماکسیمم (*min-max scaling*)» و «استانداردسازی (*standardization*)» .

هشدار

مانند سایر تخمین‌گرها، مهم است که مقیاس‌دهنده‌ها فقط با داده‌های آموزشی تطبیق داده شوند: هرگز نباید از fit() یا fit_transform() برای مجموعه‌های دیگر استفاده کنید. پس از آموزش مقیاس‌دهنده، می‌توان از آن برای transform() هر مجموعه دیگری، از جمله مجموعه اعتبارسنجی، مجموعه آزمون و داده‌های جدید، استفاده کرد. توجه داشته باشید که مقادیر مجموعه آموزشی همیشه در محدوده تعیین‌شده مقیاس‌بندی می‌شوند، اما داده‌های جدید ممکن است شامل مقادیر پرت باشند و در نتیجه خارج از این محدوده قرار گیرند. برای جلوگیری از این وضعیت، می‌توان هایپرپارامتر clip را روی True تنظیم کرد.

مقیاس‌بندی مینیمم–ماکسیمم (که اغلب «نرمال‌سازی (*Normalization*)» نامیده می‌شود) ساده‌ترین روش است: برای هر ویژگی، مقادیر جابجا و مجدداً مقیاس‌بندی می‌شوند تا در نهایت در بازه ۰ تا ۱ قرار گیرند. طبق معادله‌ی $X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$ این کار با کم کردن مقدارِ min و تقسیم بر اختلاف بین max و min انجام می‌شود. Scikit-Learn ترنسفورمری به نام MinMaxScaler

© 2025 Mohsen Zare. All rights reserved.

ارائه می‌دهد که دارای هایپرپارامتر feature_range است و به شما امکان می‌دهد در صورت نیاز، محدوده‌ای غیر از ۰ تا ۱ را انتخاب کنید (مثلاً برای شبکه‌های عصبی که ورودی‌های با «میانگین صفر (Zero-Mean Input)» بهتر عمل می‌کنند، محدوده ۱- تا ۱ ترجیح داده می‌شود). استفاده از این ترنسفورمر بسیار ساده است:

```
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
housing_num_min_max_scaled = min_max_scaler.fit_transform(housing_num)
```

«استانداردسازی» متفاوت است: ابتدا مقدار میانگین را کم می‌کند (بنابراین مقادیر استاندارد شده دارای «میانگین صفر (Zero Mean)» هستند)، سپس نتیجه را بر «انحراف معیار» تقسیم می‌کند (بنابراین مقادیر دارای انحراف معیار برابر با ۱ خواهند بود). برخلاف مقیاس‌بندی یا نرمال‌سازی مینیمم–ماکسیمم، استانداردسازی مقادیر را به یک محدوده خاص محدود نمی‌کند و کمتر تحت تأثیر «داده‌های پَرت» قرار می‌گیرد. به عنوان مثال، فرض کنید در یک منطقه به جای میانگین درآمد معمول ۰ تا ۱۵، مقدار میانگین برابر با ۱۰۰ باشد (خطای انسانی). نرمال‌سازی مین–ماکس این مقدار پرت را به مقدارِ ۱ نگاشت(map) می‌کند و سایر مقادیر را به محدوده‌ای کوچک نزدیک به صفر کاهش می‌دهد، در حالی که استانداردسازی اثر قابل توجهی روی مقادیر دیگر ندارد.

Scikit-Learn یک ترنسفورمر به نام «StandardScaler» برای اعمال استانداردسازی ارائه می‌دهد:

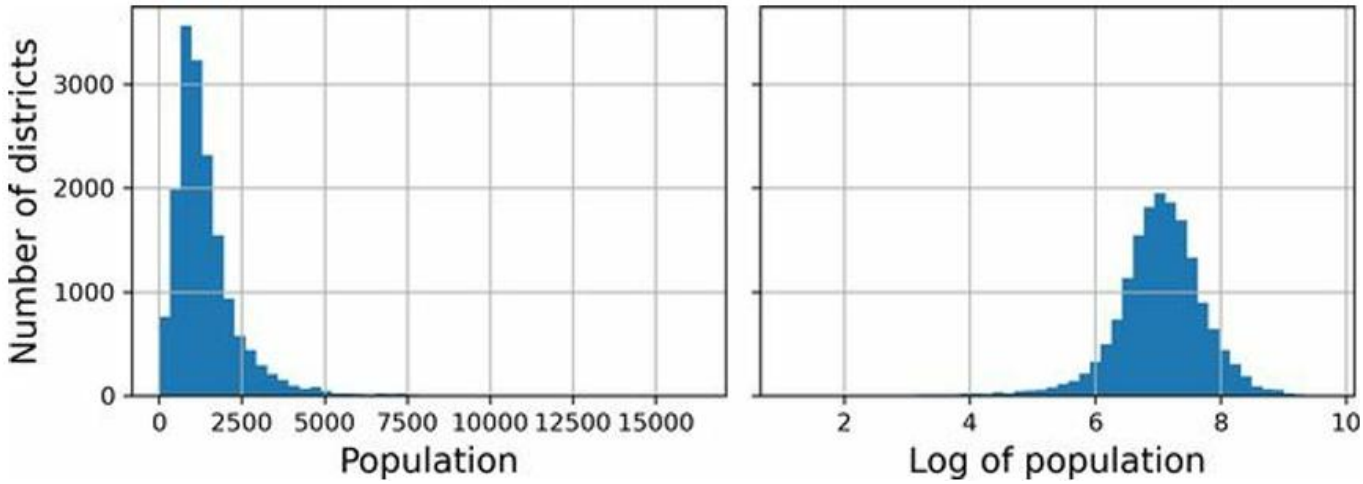
```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
housing_num_std_scaled = std_scaler.fit_transform(housing_num)
```

نکته

اگر بخواهید یک «ماتریس تُنُک (sparse matrix)» را بدون تبدیل آن به «ماتریس متراکم (dense matrix)» مقیاس‌بندی کنید، می‌توانید از StandardScaler استفاده کنید و پارامتر with_mean را روی False تنظیم کنید. در این حالت، مقادیر فقط بر انحراف معیار تقسیم می‌شوند و میانگین کم نمی‌شود، زیرا کاهش میانگین در ماتریس تُنُک باعث از بین رفتن پراکندگی داده‌ها می‌شود.

© 2025 Mohsen Zare. All rights reserved.

وقتی توزیع یک ویژگی «دنباله‌سنگین (*heavy tail*)» دارد، یعنی مقادیر دور از میانگین به صورت نَمایی(exponentially) کمیاب نیستند—به بیان دیگر، توزیع‌هایی که احتمال رخداد مقادیر بسیار بزرگ نسبتاً زیاد است؛ یعنی دم توزیع بلند و سنگین است. به این معنی که احتمال وجود مقادیر خیلی بزرگ در توزیع، بیشتر از چیزی که توزیع نرمال پیش‌بینی می‌کند —، هم مقیاس‌بندی مینیمم-ماکسیمم و هم استانداردسازی اغلب بیشتر مقادیر را در یک محدوده کوچک فشرده می‌کنند. مدل‌های یادگیری ماشین معمولاً این وضعیت را نمی‌پسندند، همان‌طور که در فصل ۴ مشاهده خواهید کرد. بنابراین، «قبل از» اعمال مقیاس‌بندی، باید ویژگی را تغییر دهید تا دنباله‌سنگین کاهش یابد و در صورت امکان توزیع تقریباً متقارن(symmetrical) شود. برای ویژگی‌های مثبت با دنباله‌سنگین در سمت راست، یک روش رایج جایگزینی ویژگی با جذر آن است (یا افزایش آن به توانی بین ۰ و ۱). اگر ویژگی دارای دنباله واقعاً بلند و سنگین باشد، مانند «توزیع قانون توان (*power law distribution*)»، استفاده از لگاریتم ویژگی می‌تواند مفید باشد. به‌عنوان مثال، ویژگی جمعیت تقریباً از قانون توان پیروی می‌کند: مناطقی با ۱۰،۰۰۰ نفر جمعیت تنها ۱۰ برابر کمتر از مناطقی با ۱،۰۰۰ نفر جمعیت فراوانی دارند، نه اینکه به صورت نمایی کمیاب باشند. شکل ۲-۱۷ نشان می‌دهد که پس از محاسبه لگاریتم این ویژگی، توزیع آن بسیار بهتر و نزدیک به «توزیع گاوسی (*Gaussian distribution*)» یا همان توزیع زنگوله‌ای شکل است.



شکل ۲-۱۷. تبدیل یک ویژگی برای نزدیک‌تر کردن آن به توزیع گاوسی

رویکرد دیگر برای مدیریت ویژگی‌های دم‌سنگین(ویژگی‌های با توزیع دم‌بلند)—یعنی داده‌های پرت زیاد یا مقادیر بسیار بزرگ دارند، دسته‌بندی آن‌ها به صورت « بازه‌بندی یا سطل‌بندی (*bucketizing*)» است که با نام‌های گسسته‌سازی یا دسته‌بندی بازه‌ای—تقسیم داده‌های پیوسته به بازه‌های گسسته (buckets) برای ساده‌تر کردن مدل‌سازی—نیز شناخته می‌شود. این روش شامل تقسیم توزیع ویژگی به بازه‌هایی تقریباً هم‌اندازه و جایگزینی هر مقدار ویژگی با «اندیس بازه‌ها» است که به آن تعلق دارد،

© 2025 Mohsen Zare. All rights reserved.

درست مانند کاری که برای ایجاد ویژگی income_cat انجام دادیم (اگرچه آنجا صرفاً برای نمونه‌گیری طبقه‌بندی‌شده استفاده شد). برای مثال، می‌توانید هر مقدار را با صدک (percentile) مربوطه جایگزین کنید. دسته‌بندی با بازه‌های مساوی منجر به تولید ویژگی‌ای با توزیع تقریباً یکنواخت می‌شود، در نتیجه نیازی به مقیاس‌بندی بیشتر نیست. همچنین می‌توانید مقادیر بازه‌ها را بر تعداد کل بازه‌ها تقسیم کنید تا مقادیر در محدوده ۰ تا ۱ قرار گیرند.

وقتی یک ویژگی دارای توزیع چندوجهی (multimodal distribution) باشد—یعنی دو یا چند قله (peak) مشخص داشته باشد که هر یک «نما (*mode*)»، یعنی مقداری که بیشترین فراوانی را در داده‌ها دارد، نامیده می‌شوند—مانند ویژگی housing_median_age، دسته‌بندی آن نیز می‌تواند مفید باشد. با این تفاوت که در این حالت، شناسه‌های بازه(bucket) باید به عنوان «دسته» در نظر گرفته شوند، نه به‌عنوان مقادیر عددی. به بیان دیگر، شاخص‌های بازه باید کدگذاری شوند، برای مثال با استفاده از OneHotEncoder (به همین دلیل معمولاً از تعداد زیادی بازه استفاده نمی‌شود). این رویکرد به مدل رگرسیون کمک می‌کند تا قوانین متفاوتی را برای محدوده‌های مختلف این ویژگی، راحت‌تر یاد بگیرد. برای مثال، ممکن است خانه‌هایی که حدود ۳۵ سال پیش ساخته شده‌اند، سبک خاصی داشته باشند که دیگر رایج نباشد، و در نتیجه ارزان‌تر از آنچه صرفاً سن آن‌ها نشان می‌دهد، باشند.

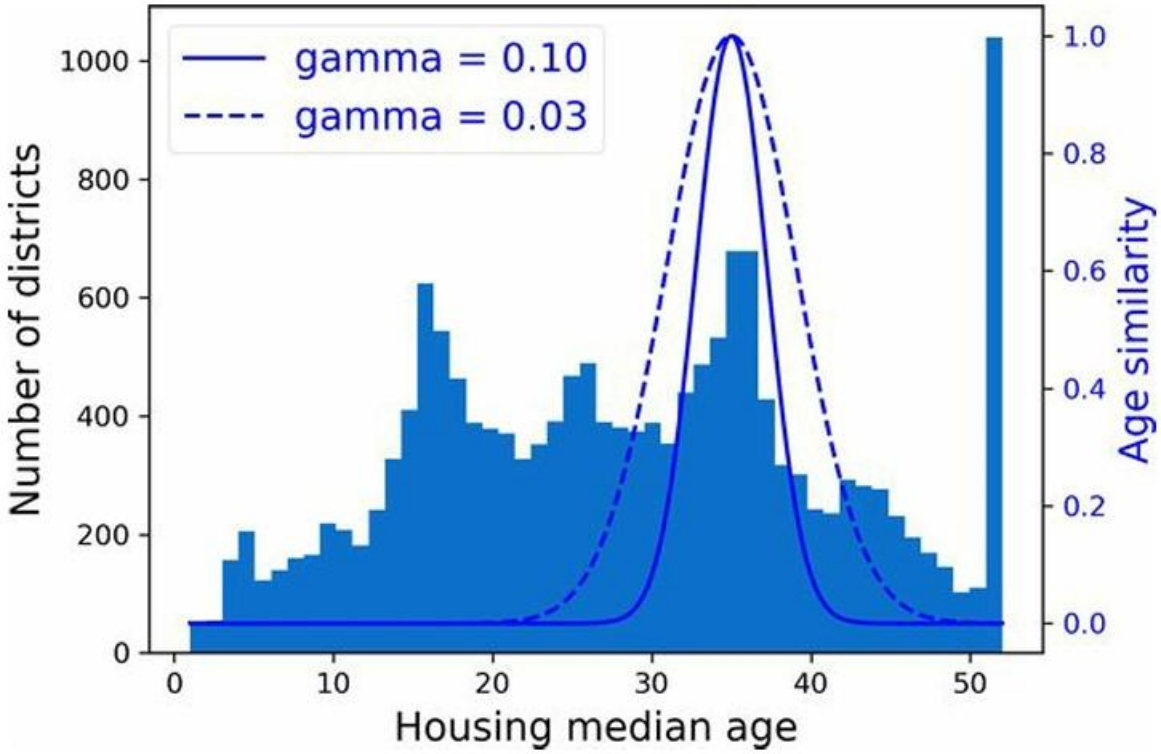
رویکرد دیگر برای تبدیل توزیع‌های چندوجهی، افزودن یک ویژگی برای هر یک از نماها (دست‌کم نماهای اصلی) است که میزان شباهت بین «میانگین سنی مسکن» و آن نمای خاص را نشان می‌دهد. معیار شباهت معمولاً با استفاده از «تابع پایه شعاعی (RBF: *radial basis function*)» محاسبه می‌شود؛ یعنی تابعی که فقط به فاصله بین مقدار ورودی و یک نقطه ثابت وابسته است. رایج‌ترین نوع RBF، «تابع پایه شعاعی گاوسی(Gaussian RBF)» است که خروجی آن با افزایش فاصله مقدار ورودی از نقطه ثابت، به‌صورت تصاعدی کاهش می‌یابد. برای مثال، شباهت بین سن مسکن x و عدد ۳۵ با استفاده از Gaussian RBF با معادلهٔ $similarity = exp(-\gamma(x - 35)^2)$ تعریف می‌شود. در این معادله، «هایپرپارامتر γ (گاما)» مشخص می‌کند که با دور شدن x از ۳۵، شباهت با چه سرعتی کاهش پیدا می‌کند. با استفاده از تابع rbf_kernel() در Scikit-Learn می‌توان یک ویژگی جدید Gaussian RBF ایجاد کرد که میزان شباهت بین میانگین سنی مسکن و ۳۵ را اندازه‌گیری می‌کند:

```
from sklearn.metrics.pairwise import rbf_kernel
```

© 2025 Mohsen Zare. All rights reserved.

```
age_simil_35 = rbf_kernel(housing[["housing_median_age"]], [[35]], gamma=0.1)
```

شکل ۲-۱۸ این ویژگی جدید را به‌عنوان تابعی از «میانگین سنی مسکن» (خط ممتد) نمایش می‌دهد و همچنین نشان می‌دهد که در صورت استفاده از مقدار γ کوچک‌تر، شکل این ویژگی چگونه تغییر می‌کند. همان‌طور که در نمودار مشخص است، ویژگی جدید «شباهت سنی» در ۳۵ سالگی به اوج خود می‌رسد، درست در حوالی قله توزیع میانگین سنی مسکن. اگر این گروه سنی خاص با قیمت‌های پایین‌تر همبستگی قابل‌توجهی داشته باشد، احتمال بالایی وجود دارد که این ویژگی جدید برای مدل مفید واقع شود.



شکل ۲-۱۸. ویژگی Gaussian RBF که میزان شباهت بین میانگین سنی مسکن و ۳۵ سال را اندازه‌گیری می‌کند.

تاکنون فقط ویژگی‌های ورودی را بررسی کرده‌ایم، اما ممکن است مقادیر هدف نیز به تبدیل نیاز داشته باشند. برای مثال، اگر توزیع هدف دارای دنباله سنگین باشد، می‌توانید مقدار هدف را با «لگاریتم (log)» آن جایگزین کنید. با این حال، در این صورت مدل رگرسیون، نه خود مقدار میانه خانه، که «لگاریتم» مقدار میانه (median) خانه را پیش‌بینی خواهد کرد. بنابراین، اگر بخواهید مقدار میانه خانه پیش‌بینی‌شده را به دست آورید، باید خروجی مدل را با تابع نمایی محاسبه کنید.

© 2025 Mohsen Zare. All rights reserved.

خوشبختانه، بیشتر ترنسفورمرهای موجود در Scikit-Learn دارای متد inverse_transform() هستند که انجام عملیات معکوس تبدیل را ساده می‌کند. به‌عنوان نمونه، در کد زیر برچسب‌ها ابتدا با استفاده از StandardScaler (مشابه کاری که برای ورودی‌ها انجام داده‌ایم) مقیاس‌بندی می‌شوند. سپس یک مدل رگرسیون خطی ساده روی این برچسب‌های مقیاس‌بندی‌شده آموزش داده می‌شود و از آن برای پیش‌بینی داده‌های جدید استفاده می‌کنیم. در ادامه، پیش‌بینی‌ها با کمک متد inverse_transform() متعلق به مقیاس‌دهنده‌ی آموزش‌دیده، دوباره به مقیاس اصلی بازگردانده می‌شوند. توجه کنید که برچسب‌ها از یک سری Pandas به یک DataFrame تبدیل می‌شوند، زیرا StandardScaler ورودی‌های دوبُعدی را دریافت می‌کند. همچنین، در این مثال برای ساده‌سازی، مدل تنها روی یک ویژگی ورودی خام یعنی میانگین درآمد آموزش داده می‌شود:

```
from sklearn.linear_model import LinearRegression
target_scaler = StandardScaler()
scaled_labels = target_scaler.fit_transform(housing_labels.to_frame())

model = LinearRegression()
model.fit(housing[["median_income"]], scaled_labels)
some_new_data = housing[["median_income"]].iloc[:5] # pretend this is new data

scaled_predictions = model.predict(some_new_data)
predictions = target_scaler.inverse_transform(scaled_predictions)
```

این روش نیز مؤثر است، اما استفاده از TransformedTargetRegressor رویکردی ساده‌تر ارائه می‌دهد. کافی است این کلاس را ایجاد کرده، مدل رگرسیون و ترنسفورمر برچسب را به آن بدهید و سپس آن را با برچسب‌های اصلیِ بدون مقیاس روی داده‌های آموزش، آموزش دهید. این ابزار به‌طور خودکار برچسب‌ها را با ترنسفورمر مشخص‌شده مقیاس‌بندی کرده و مدل رگرسیون را بر اساس برچسب‌های مقیاس‌بندی‌شده آموزش می‌دهد، مشابه فرآیندی که پیش‌تر انجام دادیم. در زمان پیش‌بینی نیز، ابتدا متد predict() مدل رگرسیون را فراخوانی می‌کند و سپس خروجی را با استفاده از متد inverse_transform() ترنسفورمر، به مقیاس اصلی بازمی‌گرداند:

```
from sklearn.compose import TransformedTargetRegressor

model = TransformedTargetRegressor(LinearRegression(),
                    transformer=StandardScaler())
model.fit(housing[["median_income"]], housing_labels)
predictions = model.predict(some_new_data)
```

© 2025 Mohsen Zare. All rights reserved.

ترنسفورمرهای سفارشی

اگرچه Scikit-Learn مجموعه‌ای از ترنسفورمرهای آماده و کاربردی را ارائه می‌دهد، اما در مواردی مانند انجام تبدیل‌های خاص، پاک‌سازی‌های سفارشی یا ترکیب ویژگی‌های ویژه، لازم است ترنسفورمرهای اختصاصی خود را پیاده‌سازی کنید. برای تبدیل‌هایی که نیاز به فرایند آموزش ندارند، می‌توان یک تابع نوشت که یک آرایه NumPy را به‌عنوان ورودی دریافت کرده و نسخه تبدیل‌شده آن را به‌عنوان خروجی بازگرداند. به‌عنوان نمونه، همان‌طور که در بخش پیشین اشاره شد، یکی از رویکردهای رایج برای ویژگی‌هایی با توزیع دنباله‌سنگین (به‌ویژه وقتی دم در سمت راست و ویژگی مثبت است) جایگزینی مقدار آن‌ها با لگاریتمشان است. اکنون یک «ترنسفورمر لگاریتمی» طراحی می‌کنیم و آن را بر روی ویژگی population اعمال می‌کنیم.:

```
from sklearn.preprocessing import FunctionTransformer

log_transformer = FunctionTransformer(np.log, inverse_func=np.exp)
log_pop = log_transformer.transform(housing[["population"]])
```

آرگومان inverse_func یک گزینه اختیاری است که امکان مشخص کردن یک تابع معکوس برای تبدیل را فراهم می‌کند. این قابلیت به ویژه زمانی مفید است که بخواهید ترنسفورمر سفارشی خود را در ترکیب با TransformedTargetRegressor به کار ببرید، زیرا این کلاس هم در مرحله آموزش و هم پیش‌بینی، نیازمند انجام تبدیل و سپس بازگرداندن مقادیر به مقیاس اصلی است.

همچنین، تابع تبدیل سفارشی شما می‌تواند هایپرپارامترها را به عنوان آرگومان‌های اضافی دریافت کند تا رفتار آن قابل تنظیم باشد. برای نمونه، در ادامه نحوه پیاده‌سازی یک ترنسفورمر که همان معیار شباهت Gaussian RBF را محاسبه می‌کند، مشابه آنچه پیش‌تر دیدیم، آورده شده است:

```
rbf_transformer = FunctionTransformer(rbf_kernel,
                    kw_args=dict(Y=[[35.]], gamma=0.1))
age_simil_35 = rbf_transformer.transform(housing[["housing_median_age"]])
```

توجه داشته باشید که هیچ تابع معکوسی برای kernel یا هسته‌ی پردازشی RBF وجود ندارد، زیرا در فاصله مشخص از یک نقطه ثابت، همیشه بیش از یک مقدار وجود دارد (به جز فاصله صفر).

© 2025 Mohsen Zare. All rights reserved.

همچنین باید در نظر داشته باشید که rbf_kernel() به صورت مستقل روی هر ویژگی عمل نمی‌کند؛ اگر آرایه‌ای شامل دو ویژگی به آن داده شود، فاصله دوبُعدی یا اُقلیدُسی (Euclidean) برای محاسبه شباهت استفاده می‌شود. برای مثال، در ادامه نحوه افزودن یک ویژگی جدید که شباهت جغرافیایی هر منطقه را با سانفرانسیسکو اندازه‌گیری می‌کند، نشان داده شده است:

```
sf_coords = 37.7749, -122.41
sf_transformer = FunctionTransformer(rbf_kernel,
                    kw_args=dict(Y=[sf_coords], gamma=0.1))
sf_simil = sf_transformer.transform(housing[["latitude", "longitude"]])
```

ترنسفورمرهای سفارشی برای ترکیب ویژگی‌ها نیز کاربردی هستند. برای مثال، در اینجا یک FunctionTransformer نشان داده شده است که نسبت بین ویژگی‌های ورودی ۰ و ۱ را محاسبه می‌کند:

```
>>> ratio_transformer = FunctionTransformer(lambda X: X[:, [0]] / X[:, [1]])
>>> ratio_transformer.transform(np.array([[1., 2.], [3., 4.]]))
array([[0.5 ],
       [0.75]])
```

FunctionTransformer بسیار کاربردی است، اما اگر بخواهید ترنسفورمر شما قابلیت آموزش داشته باشد، یعنی برخی پارامترها را در متد fit() بیاموزد و سپس در متد transform() از آن‌ها استفاده کند، چه باید کرد؟ در این صورت، باید یک کلاس سفارشی ایجاد کنید. Scikit-Learn از نوع‌دهی اردکی (duck typing) استفاده می‌کند—اشاره به این ایده: «اگر چیزی مثل اردک راه میرود و مثل اردک صدا میدهد، پس اردک است!»؛ یعنی نوع (type) داده بر اساس رفتار یا ویژگی‌ها تشخیص داده میشود نه تعریف صریح؛ بنابراین این کلاس نیازی به ارث‌بری از هیچ کلاس پایه خاصی ندارد. تنها شرط، پیاده‌سازی سه متد fit()، transform() و fit_transform() است، به‌طوری که fit() باید self را بازگرداند.

شما می‌توانید با اضافه کردن TransformerMixin به عنوان کلاس پایه، متد fit_transform() را به صورت پیش‌فرض دریافت کنید؛ پیاده‌سازی پیش‌فرض این متد، ابتدا fit() و سپس transform() را فراخوانی می‌کند. اگر BaseEstimator را نیز به عنوان کلاس پایه اضافه کنید (و از *args و

© 2025 Mohsen Zare. All rights reserved.

**kwargs در constructor استفاده نکنید)، دو متد دیگر نیز در اختیار شما قرار می‌گیرد: get_params() و set_params(). این متدها برای تنظیم خودکار هایپرپارامترها بسیار مفید هستند.

برای مثال، در ادامه یک ترنسفورمر سفارشی نشان داده شده که عملکردی مشابه StandardScaler دارد:

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils.validation import check_array, check_is_fitted

class StandardScalerClone(BaseEstimator, TransformerMixin):
  def __init__(self, with_mean=True): # no *args or **kwargs!
    self.with_mean = with_mean
  def fit(self, X, y=None): # y is required even though we don't use it
    X = check_array(X) # checks that X is an array with finite float values
    self.mean_ = X.mean(axis=0)
    self.scale_ = X.std(axis=0)
    self.n_features_in_ = X.shape[1] # every estimator stores this in fit()
    return self # always return self!
  def transform(self, X):
  check_is_fitted(self) # looks for learned attributes (with trailing _)
  X = check_array(X)
  assert self.n_features_in_ == X.shape[1]
  if self.with_mean:
    X = X - self.mean_
  return X / self.scale_
```

در اینجا چند نکته مهم وجود دارد:

- پکیجِ sklearn.utils.validation شامل چندین تابع است که می‌توان برای اعتبارسنجی ورودی‌ها از آن‌ها استفاده کرد. برای ساده‌سازی، در ادامه کتاب از چنین آزمون‌هایی صرف نظر شده است، اما در کد عملیاتی واقعی باید این اعتبارسنجی‌ها لحاظ شوند.
- Pipeline در Scikit-Learn به متد fit() نیاز دارند که دو آرگومان X و y دریافت کند. به همین دلیل است که معمولاً آرگومان y=None اضافه می‌شود، حتی اگر از y استفاده نکنیم.

© 2025 Mohsen Zare. All rights reserved.

- همه تخمین‌گرهای Scikit-Learn مقدار n_features_in_ را در متد fit() تنظیم می‌کنند و تضمین می‌کنند که داده‌های ارسالی به transform() یا predict() همان تعداد ویژگی را داشته باشند.
- متد fit() باید مقدار self را برگرداند.
- این پیاده‌سازی صددرصد کامل نیست: همه تخمین‌گرها باید هنگام دریافت یک DataFrame در متد fit()، feature_names_in_ را تنظیم کنند. همچنین، همه ترنسفورمرها باید در صورت قابل معکوس بودن تبدیل، متد get_feature_names_out() و inverse_transform() را ارائه دهند. برای جزئیات بیشتر می‌توانید به آخرین تمرین فصل مراجعه کنید.

یک ترنسفورمر سفارشی می‌تواند از تخمین‌گرهای دیگر در پیاده‌سازی خود استفاده کند و این کار اغلب مفید است. برای مثال، می‌توان ترنسفورمری طراحی کرد که در متد fit() از خوشه‌بند KMeans برای شناسایی خوشه‌های اصلی در داده‌های آموزشی بهره ببرد و سپس در متد transform() از rbf_kernel() برای اندازه‌گیری میزان شباهت هر نمونه به مراکز خوشه استفاده کند:

```
from sklearn.cluster import KMeans

class ClusterSimilarity(BaseEstimator, TransformerMixin):
  def __init__(self, n_clusters=10, gamma=1.0, random_state=None):
    self.n_clusters = n_clusters
    self.gamma = gamma
    self.random_state = random_state
  def fit(self, X, y=None, sample_weight=None):
    self.kmeans_ = KMeans(self.n_clusters, random_state=self.random_state)
    self.kmeans_.fit(X, sample_weight=sample_weight)
    return self  # always return self!
  def transform(self, X):
    return rbf_kernel(X, self.kmeans_.cluster_centers_, gamma=self.gamma)
  def get_feature_names_out(self, names=None):
    return [f"Cluster {i} similarity" for i in range(self.n_clusters)]
```

© 2025 Mohsen Zare. All rights reserved.

نکته

می‌توانید با ارسال یک نمونه به تابع check_estimator() از پکیج sklearn.utils.estimator_checks بررسی کنید که آیا تخمین‌گر سفارشی شما با API سایکیت‌لرن سازگار است یا خیر. برای مشاهده مستندات کامل API ، به آدرس *https://scikit-learn.org/stable/developers* مراجعه کنید.

همان‌طور که در فصل ۹ خواهید دید، k-means یک الگوریتم خوشه‌بندی است که خوشه‌ها را در داده‌ها شناسایی می‌کند. تعداد خوشه‌هایی که الگوریتم به دنبال آن‌ها است، توسط پارامتر n_clusters تعیین می‌شود. پس از آموزش، مراکز خوشه‌ها از طریق ویژگی cluster_centers_ در دسترس قرار دارند. متد fit() در KMeans یک آرگومان اختیاری به نام sample_weight دارد که به کاربر اجازه می‌دهد وزن نسبی نمونه‌ها را مشخص کند. الگوریتم k-means یک الگوریتم تصادفی است، به این معنا که برای تعیین موقعیت خوشه‌ها به تصادف وابسته است؛ بنابراین، برای داشتن نتایج تکرارپذیر، باید پارامتر random_state را تنظیم کنید. همان‌طور که می‌بینید، با وجود پیچیدگی مفهومی، کد نسبتاً ساده و قابل‌فهم است. حالا بیایید این ترانسفورمر سفارشی را به کار بگیریم:

```
cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
similarities = cluster_simil.fit_transform(housing[["latitude", "longitude"]],
                                           sample_weight=housing_labels)
```

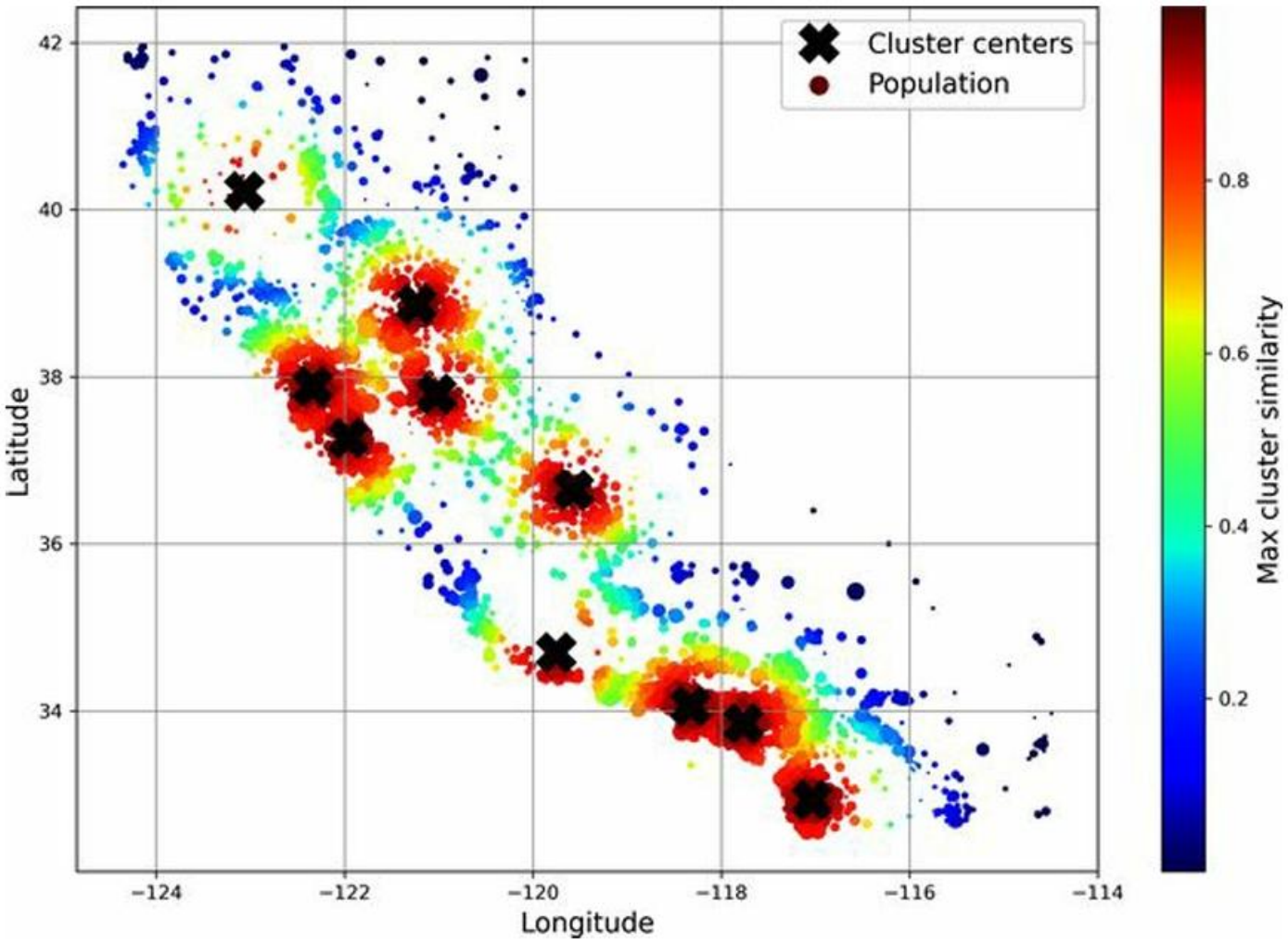
این کد یک ترنسفورمر ClusterSimilarity ایجاد می‌کند و تعداد خوشه‌ها را روی ۱۰ تنظیم می‌کند. سپس متد fit_transform() را روی طول و عرض جغرافیایی هر منطقه در مجموعه‌ی آموزشی فراخوانی می‌کند و به هر منطقه بر اساس میانگین ارزش خانه‌های آن، وزن می‌دهد. این ترانسفورمر از k-means برای تعیین موقعیت خوشه‌ها استفاده می‌کند و سپس شباهت Gaussian RBF بین هر منطقه و هر ۱۰ مرکز خوشه را محاسبه می‌کند. خروجی، ماتریسی است که هر ردیف آن متناظر با یک منطقه و هر ستون آن متناظر با یک خوشه است. در ادامه، سه ردیف اول ماتریس را گرد شده تا دو رقم اعشار مشاهده می‌کنیم:

```
>>> similarities[:3].round(2)
```

© 2025 Mohsen Zare. All rights reserved.

```
array([[0.  , 0.14, 0.  , 0.  , 0.  , 0.08, 0.  , 0.99, 0.  , 0.6 ],
       [0.63, 0.  , 0.99, 0.  , 0.  , 0.  , 0.04, 0.  , 0.11, 0.  ],
       [0.  , 0.29, 0.  , 0.  , 0.01, 0.44, 0.  , 0.7 , 0.  , 0.3 ]])
```

شکل ۲-۱۹ ده مرکز خوشه‌ای را که توسط k-means یافته شده‌اند، نشان می‌دهد. مناطق براساس شباهت جغرافیایی‌شان به نزدیک‌ترین مرکز خوشه، رنگ‌آمیزی شده‌اند. همان‌طور که دیده می‌شود، بیشتر خوشه‌ها در مناطق پرجمعیت و با ارزش مسکن بالاتر قرار دارند.



شکل ۲-۱۹ شباهت هر منطقه به نزدیک‌ترین مرکز خوشه را با استفاده از تابع پایه شعاعی گاوسی (Gaussian RBF) نشان می‌دهد.

© 2025 Mohsen Zare. All rights reserved.

پایپ‌لاین‌های ترنسفورمر

همان‌طور که مشاهده می‌کنید، مراحل تبدیل داده متعدد هستند و باید به ترتیب درست اجرا شوند. خوشبختانه، Scikit-Learn کلاس pipeline را برای مدیریت چنین توالی‌هایی از تبدیل‌ها ارائه می‌دهد. در ادامه، نمونه‌ای از یک pipeline کوچک برای ویژگی‌های عددی آمده است که ابتدا ویژگی‌های ورودی را دریافت می‌کند و سپس آن‌ها را مقیاس‌بندی می‌کند:

```
from sklearn.pipeline import Pipeline

num_pipeline = Pipeline([
  ("impute", SimpleImputer(strategy="median")),
  ("standardize", StandardScaler()),
])
```

سازنده‌ی pipeline یک فهرست از جفت‌های نام-تخمین‌گر (تاپل دو تایی) دریافت می‌کند که ترتیب اجرای مراحل را مشخص می‌کنند. نام‌ها می‌توانند هر مقداری باشند، به شرطی که یکتا باشند و شامل دو زیرخط (__) نباشند. این نام‌ها بعدها در تنظیم هایپرپارامترها کاربرد خواهند داشت. تمامی تخمین‌گرها باید ترنسفورمر باشند، یعنی متد fit_transform() را داشته باشند، به جز آخرین مورد که می‌تواند یک ترنسفورمر، پیش‌بینی‌کننده یا هر نوع تخمین‌گر دیگری باشد.

> **نکته**
>
> در یک نوت‌بوک ژوپیتر، اگر کتابخانه‌ی sklearn را وارد (import) کنید و دستور sklearn.set_config(display="diagram") را اجرا نمایید، تمام تخمین‌گرهای Scikit-Learn به صورت نمودارهای تعاملی نمایش داده می‌شوند. این قابلیت برای تجسم و نمایش pipeline بسیار مفید است. برای مشاهده‌ی نمودار num_pipeline، کافی است یک سلول ایجاد کرده و num_pipeline را به‌عنوان آخرین خط آن سلول قرار دهید. با کلیک روی هر تخمین‌گر، جزئیات بیشتری درباره‌ی آن نمایش داده خواهد شد.

اگر نیازی به نامگذاری دستی ترنسفورمرها ندارید، می‌توانید از تابع make_pipeline() استفاده کنید.

© 2025 Mohsen Zare. All rights reserved.

این تابع ترنسفورمرها را به‌عنوان آرگومان‌های موقعیتی دریافت می‌کند و یک Pipeline ایجاد می‌کند که نام هر مرحله بر اساس نام کلاس ترنسفورمر با حروف کوچک و بدون زیرخط تنظیم می‌شود (مثلاً "simpleimputer"):

```
from sklearn.pipeline import make_pipeline

num_pipeline = make_pipeline(SimpleImputer(strategy="median"), StandardScaler())
```

اگر چندین ترنسفورمر نام یکسانی داشته باشند، یک اندیس به نام آنها اضافه می‌شود تا یکتا شوند (مثلاً foo-1، foo-2 و غیره).

وقتی متد fit() در یک pipeline فراخوانی می‌شود، به صورت متوالی fit_transform() روی همه ترنسفورمرها اجرا می‌شود و خروجی هر مرحله به مرحله بعدی داده می‌شود. در نهایت، وقتی به تخمین‌گر نهایی می‌رسیم، تنها متد fit() روی آن فراخوانی می‌شود.

این pipeline متدهای تخمین‌گر نهایی را نیز در دسترس قرار می‌دهد. در این مثال، آخرین تخمین‌گر یک StandardScaler است که یک ترنسفورمر محسوب می‌شود، بنابراین کل pipeline مانند یک ترنسفورمر عمل می‌کند. اگر متد transform() از pipeline فراخوانی شود، تمام تبدیل‌ها به ترتیب روی داده‌ها اعمال می‌شوند. اما اگر آخرین تخمین‌گر یک پیش‌بین باشد، pipeline به جای transform()، متد predict() ارائه می‌دهد و فراخوانی آن، ابتدا تمام تبدیل‌ها را به ترتیب اجرا کرده و سپس نتیجه را به predict() پیش‌بینی‌کننده منتقل می‌کند.

بیایید متد fit_transform() را روی pipeline فراخوانی کنیم و دو ردیف اول خروجی آن را، پس از گرد کردن تا دو رقم اعشار، بررسی کنیم:

```
>>> housing_num_prepared = num_pipeline.fit_transform(housing_num)
>>> housing_num_prepared[:2].round(2)
array([[-1.42, 1.01, 1.86, 0.31, 1.37, 0.14, 1.39, -0.94],
       [ 0.6 , -0.7 , 0.91, -0.31, -0.44, -0.69, -0.37, 1.17]])
```

همانطور که پیش‌تر مشاهده شد، برای recall یک DataFrame با ظاهر مرتب، می‌توانید از متد get_feature_names_out() در pipeline استفاده کنید:

© 2025 Mohsen Zare. All rights reserved.

```
df_housing_num_prepared = pd.DataFrame(
  housing_num_prepared, columns=num_pipeline.get_feature_names_out(),
  index=housing_num.index)
```

پایپ‌لاین از اندیس‌گذاری پشتیبانی می‌کنند؛ برای مثال، pipeline[1] دومین تخمین‌گر در پایپ‌لاین را برمی‌گرداند، و pipeline[:-1] یک آبجکت پایپ‌لاین شامل همه تخمین‌گرها به جز آخرین تخمین‌گر ایجاد می‌کند. همچنین می‌توانید از طریق ویژگی steps که فهرستی از جفت‌های نام-تخمین‌گر است، یا از طریق ویژگی دیکشنری named_steps که نام‌ها را به تخمین‌گرها نگاشت می‌کند، به تخمین‌گرها دسترسی داشته باشید. برای نمونه، num_pipeline["simpleimputer"] تخمین‌گری با نام "simpleimputer" را بازمی‌گرداند.

تاکنون ستون‌های دسته‌بندی‌شده و ستون‌های عددی را جداگانه مدیریت کرده‌ایم. اما ساده‌تر است که یک ترنسفورمر واحد داشته باشیم که بتواند همه ستون‌ها را مدیریت کرده و تبدیل‌های مناسب را روی هر ستون اعمال کند. برای این منظور، می‌توان از ColumnTransformer استفاده کرد. به عنوان مثال، ColumnTransformer زیر، num_pipeline (همان‌که قبلاً تعریف شد) را روی ویژگی‌های عددی و cat_pipeline را روی ویژگی‌های دسته‌بندی اعمال می‌کند:

```
from sklearn.compose import ColumnTransformer

num_attribs = ["longitude", "latitude", "housing_median_age", "total_rooms",
        "total_bedrooms", "population", "households", "median_income"]
cat_attribs = ["ocean_proximity"]

cat_pipeline = make_pipeline(
  SimpleImputer(strategy="most_frequent"),
  OneHotEncoder(handle_unknown="ignore"))

preprocessing = ColumnTransformer([
  ("num", num_pipeline, num_attribs),
  ("cat", cat_pipeline, cat_attribs),
])
```

© 2025 Mohsen Zare. All rights reserved.

ابتدا کلاس ColumnTransformer را وارد می‌کنیم، سپس لیست نام‌های ستون‌های عددی و دسته‌بندی‌شده را مشخص می‌کنیم و یک pipeline ساده برای ویژگی‌های دسته‌بندی‌شده می‌سازیم. در نهایت، یک ColumnTransformer ایجاد می‌کنیم. سازنده‌ی آن به یک لیست از سه‌تایی‌ها (3-tuples) نیاز دارد که هر سه‌تایی شامل یک نام (که باید منحصر به فرد باشد و شامل دو زیرخط نباشد)، یک ترنسفورمر و لیستی از نام‌ها یا اندیس‌های ستون‌هایی است که ترنسفورمر باید روی آن‌ها اعمال شود.

> **نکته**
>
> به جای استفاده از یک ترنسفورمر، می‌توانید رشته "drop" را مشخص کنید تا ستون‌ها حذف شوند، یا رشته "passthrough" را مشخص کنید تا ستون‌ها بدون تغییر باقی بمانند. به طور پیش‌فرض، ستون‌های باقی‌مانده (یعنی آن‌هایی که در لیست سه‌تایی‌ها نیستند) حذف می‌شوند، اما اگر بخواهید این ستون‌ها به شکل متفاوت مدیریت شوند، می‌توانید هایپرپارامتر remainder را روی یک ترنسفورمر یا "passthrough" تنظیم کنید.

از آنجایی که فهرست کردن تمام نام ستون‌ها دشوار است، Scikit-Learn تابع make_column_selector() را ارائه می‌دهد که یک تابع انتخابگر بازمی‌گرداند و می‌توانید از آن برای انتخاب خودکار تمام ویژگی‌های یک نوع داده مشخص، مانند عددی یا دسته‌بندی، استفاده کنید. این تابع انتخابگر را می‌توان به جای نام ستون‌ها یا اندیس‌ها به ColumnTransformer ارسال کرد. علاوه بر این، اگر نامگذاری ترنسفورمرها برای شما اهمیت ندارد، می‌توانید از make_column_transformer() استفاده کنید که نام‌ها را به طور خودکار انتخاب می‌کند، مشابه make_pipeline(). برای مثال، کد زیر همان ColumnTransformer قبلی را ایجاد می‌کند، با این تفاوت که ترنسفورمرها به جای "num" و "cat"، به صورت "pipeline-1" و "pipeline-2" نامگذاری می‌شوند:

```
from sklearn.compose import make_column_selector, make_column_transformer

preprocessing = make_column_transformer(
  (num_pipeline, make_column_selector(dtype_include=np.number)),
```

© 2025 Mohsen Zare. All rights reserved.

```
  (cat_pipeline, make_column_selector(dtype_include=object)),
)
```

اکنون آماده‌ایم تا این ColumnTransformer را روی داده‌های housing اعمال کنیم و تمام تبدیل‌های عددی و دسته‌بندی را به صورت یکپارچه اجرا کنیم:

```
housing_prepared = preprocessing.fit_transform(housing)
```

عالی! اکنون یک pipeline کامل پیش‌پردازش داریم که کل مجموعه داده‌های آموزشی را دریافت می‌کند، تمام ترنسفورمرها را روی ستون‌های مناسب اعمال می‌کند و سپس ستون‌های تبدیل‌شده را به صورت افقی به هم متصل می‌کند (توجه داشته باشید که ترنسفورمرها هرگز نباید تعداد ردیف‌ها را تغییر دهند). خروجی این pipeline، یک آرایه NumPy است، اما با استفاده از متد get_feature_names_out() از آبجکت preprocessing می‌توانید نام ستون‌ها را دریافت کرده و داده‌ها را در قالب یک DataFrame زیبا نمایش دهید.

نکته

تابع OneHotEncoder یک ماتریس تُنُک و تابع num_pipeline یک ماتریس متراکم برمی‌گردانند. وقتی چنین ترکیبی از ماتریس‌های پراکنده و متراکم وجود داشته باشد، ColumnTransformer چگالی ماتریس نهایی (نسبت سلول‌های غیر صفر) را تخمین می‌زند و اگر این چگالی کمتر از یک آستانه مشخص باشد (به طور پیش‌فرض sparse_threshold=0.3)، یک ماتریس تُنُک برمی‌گرداند. در این مثال، خروجی یک ماتریس متراکم است.

پروژه شما بسیار خوب پیش می‌رود و تقریباً آماده هستید تا چند مدل یادگیری ماشین را آموزش دهید. در این مرحله می‌خواهید یک pipeline واحد بسازید که تمام تبدیل‌هایی را که تاکنون آزمایش کرده‌اید، انجام دهد. در ادامه، خلاصه‌ای از کارکرد این pipeline و دلایل آن آمده است:

© 2025 Mohsen Zare. All rights reserved.

- مقادیر ناموجود در ویژگی‌های عددی توسط جایگزینی با «میانه» پر می‌شوند، زیرا اکثر الگوریتم‌های یادگیری ماشین (ML) نمی‌توانند با مقادیر ناموجود کار کنند. در ویژگی‌های دسته‌بندی، مقادیر ناموجود با رایج‌ترین دسته‌بندی جایگزین می‌شوند.

- ویژگی‌های دسته‌بندی‌شده به صورت one-hot کدگذاری می‌شوند، زیرا اکثر الگوریتم‌های یادگیری ماشین فقط ورودی‌های عددی را می‌پذیرند.

- چند ویژگی نسبت (ratio) جدید محاسبه و اضافه می‌شوند: نسبت اتاق‌خواب‌ها، تعداد اتاق‌ها در هر خانه و تعداد افراد در هر خانه. این ویژگی‌ها احتمالاً همبستگی بهتری با ارزش متوسط خانه دارند و به مدل کمک می‌کنند.

- چند ویژگی شباهت خوشه‌ای اضافه می‌شوند که معمولاً برای مدل مفیدتر از طول و عرض جغرافیایی خواهند بود.

- ویژگی‌هایی با دنباله‌سنگین با لگاریتم جایگزین می‌شوند، زیرا اکثر مدل‌ها ویژگی‌هایی همراه با توزیع تقریباً یکنواخت یا گاوسی را ترجیح می‌دهند.

- تمام ویژگی‌های عددی استانداردسازی می‌شوند، زیرا اکثر الگوریتم‌های یادگیری ماشین ترجیح می‌دهند همه ویژگی‌ها تقریباً در یک مقیاس باشند.

کدی که برای ساخت pipeline به‌منظور انجام تمام تبدیل‌های گفته‌شده استفاده می‌شود، تا این مرحله باید برای شما آشنا باشد:

```
def column_ratio(X):
  return X[:, [0]] / X[:, [1]]

def ratio_name(function_transformer, feature_names_in):
  return ["ratio"] # feature names out

def ratio_pipeline():
  return make_pipeline(
    SimpleImputer(strategy="median"),
    FunctionTransformer(column_ratio, feature_names_out=ratio_name),
    StandardScaler())
log_pipeline = make_pipeline(
```

© 2025 Mohsen Zare. All rights reserved.

```
  SimpleImputer(strategy="median"),
  FunctionTransformer(np.log, feature_names_out="one-to-one"),
  StandardScaler())
cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
default_num_pipeline = make_pipeline(SimpleImputer(strategy="median"),
                    StandardScaler())
preprocessing = ColumnTransformer([
    ("bedrooms", ratio_pipeline(), ["total_bedrooms", "total_rooms"]),
    ("rooms_per_house", ratio_pipeline(), ["total_rooms", "households"]),
    ("people_per_house", ratio_pipeline(), ["population", "households"]),
    ("log", log_pipeline, ["total_bedrooms", "total_rooms", "population",
                  "households", "median_income"]),
    ("geo", cluster_simil, ["latitude", "longitude"]),
    ("cat", cat_pipeline, make_column_selector(dtype_include=object)),
],
remainder=default_num_pipeline) # one column remaining: housing_median_age
```

اگر این ColumnTransformer را اجرا کنید، تمام تبدیل‌ها اعمال شده و یک آرایه NumPy شامل ۲۴ ویژگی به‌عنوان خروجی تولید می‌شود:

```
>>> housing_prepared = preprocessing.fit_transform(housing)
>>> housing_prepared.shape
(16512, 24)
>>> preprocessing.get_feature_names_out()
array(['bedrooms__ratio', 'rooms_per_house__ratio',
       'people_per_house__ratio', 'log__total_bedrooms',
       'log__total_rooms', 'log__population', 'log__households',
       'log__median_income', 'geo__Cluster 0 similarity', [...],
       'geo__Cluster 9 similarity', 'cat__ocean_proximity_<1H OCEAN',
       'cat__ocean_proximity_INLAND', 'cat__ocean_proximity_ISLAND',
       'cat__ocean_proximity_NEAR BAY', 'cat__ocean_proximity_NEAR OCEAN',
       'remainder__housing_median_age'], dtype=object)
```

© 2025 Mohsen Zare. All rights reserved.

انتخاب و آموزش یک مدل

پس از اینکه مسئله را چارچوب‌بندی کردید، داده‌ها را بررسی و دریافت نمودید، مجموعه‌های آموزشی و آزمون را نمونه‌گیری کردید و یک pipeline پیش‌پردازش طراحی کردید که داده‌ها را برای الگوریتم‌های یادگیری ماشین به‌طور خودکار پاکسازی و آماده می‌کند، اکنون آماده هستید تا یک مدل یادگیری ماشین مناسب را انتخاب و آموزش دهید.

© 2025 Mohsen Zare. All rights reserved.

آموزش و ارزیابی روی مجموعه آموزشی

خبر خوب این است که به دلیل تمام مراحل آماده‌سازی قبلی، کار بسیار ساده شده است. برای شروع، می‌توانید یک مدل رگرسیون خطی ساده را آموزش دهید:

```
from sklearn.linear_model import LinearRegression

lin_reg = make_pipeline(preprocessing, LinearRegression())
lin_reg.fit(housing, housing_labels)
```

تمام شد! حالا شما یک مدل رگرسیون خطی کارآمد دارید. آن را روی مجموعه آموزشی امتحان می‌کنید، به پنج پیش‌بینی اول نگاه می‌کنید و آنها را با برچسب‌ها مقایسه می‌کنید:

```
>>> housing_predictions = lin_reg.predict(housing)
>>> housing_predictions[:5].round(-2)  # -2 = rounded to the nearest hundred
array([243700., 372400., 128800.,  94400., 328300.])
>>> housing_labels.iloc[:5].values
array([458300., 483800., 101700.,  96100., 361800.])
```

این روش عملکرد قابل قبولی دارد، اما همیشه دقیق نیست: پیش‌بینی اول بیش از ۲۰۰,۰۰۰ دلار خطا دارد، در حالی که سایر پیش‌بینی‌ها بهتر هستند؛ دو مورد حدود ۲۵% و دو مورد دیگر کمتر از ۱۰% خطا دارند. به یاد داشته باشید که معیار عملکرد شما RMSE است، بنابراین باید RMSE مدل رگرسیون را روی کل مجموعه‌ی آموزشی با استفاده از تابع mean_squared_error() از Scikit-Learn محاسبه کنید و آرگومان squared را روی False تنظیم نمایید:

```
>>> from sklearn.metrics import mean_squared_error
>>> lin_rmse = mean_squared_error(housing_labels, housing_predictions,
 ...                               squared=False)
 ...
>>> lin_rmse
68687.89176589991
```

این مدل از هیچی بهتر است، اما هنوز عملکرد رضایت‌بخشی ندارد: میانگین ارزش خانه‌ها در اکثر مناطق بین ۱۲۰,۰۰۰ تا ۲۶۵,۰۰۰ دلار است و خطای پیش‌بینی معمول ۶۸,۶۲۸ دلار نشان می‌دهد که

© 2025 Mohsen Zare. All rights reserved.

مدل به خوبی برازش نشده است. این وضعیت نمونه‌ای از «underfitting» است، یعنی مدل داده‌های آموزشی را کمتر از حد لازم یاد گرفته است. چنین وضعیتی می‌تواند به این معنا باشد که ویژگی‌ها اطلاعات کافی برای پیش‌بینی دقیق ارائه نمی‌دهند یا مدل به اندازه کافی قدرتمند نیست. همان‌طور که در فصل قبل دیدیم، راه‌های اصلی رفع کم‌برازش شامل انتخاب یک مدل قوی‌تر، تغذیه الگوریتم با ویژگی‌های بهتر یا کاهش محدودیت‌های مدل است. از آنجا که این مدل منظم نشده است، گزینه‌ی آخر منتفی است. می‌توانید ویژگی‌های بیشتری اضافه کنید، اما ابتدا بهتر است یک مدل پیچیده‌تر را آزمایش کنید تا عملکرد آن را ارزیابی نمایید.

شما تصمیم می‌گیرید از DecisionTreeRegressor استفاده کنید، زیرا این مدل نسبتاً قدرتمند است و توانایی شناسایی روابط غیرخطی پیچیده در داده‌ها را دارد. (درخت‌های تصمیم‌گیری با جزئیات بیشتر در فصل ۶ بررسی شده‌اند):

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))
tree_reg.fit(housing, housing_labels)
```

پس از آموزش مدل، آن را روی مجموعه آموزشی ارزیابی می‌کنید:

```
>>> housing_predictions = tree_reg.predict(housing)
>>> tree_rmse = mean_squared_error(housing_labels, housing_predictions,
...                                squared=False)
...
>>> tree_rmse
0.0
```

صبر کنید، چی؟ هیچ خطایی وجود ندارد؟ آیا واقعاً ممکن است مدلی کاملاً بدون خطا باشد؟ احتمال بیشتری وجود دارد که مدل به شدت بیش‌برازش (overfitting) شده باشد. برای اطمینان، همان‌طور که پیش‌تر آموختید، تا زمانی که مدل نهایی آماده نشده است، نباید از مجموعه آزمون استفاده کنید. بنابراین لازم است بخشی از مجموعه آموزشی را برای آموزش و بخشی دیگر را برای اعتبارسنجی مدل کنار بگذارید.

© 2025 Mohsen Zare. All rights reserved.

ارزیابی بهتر با استفاده از اعتبارسنجی متقاطع

برای ارزیابی بهتر مدل درخت تصمیم، می‌توان ابتدا با استفاده از تابع train_test_split() مجموعه آموزشی را به دو بخش تقسیم کرد: یک مجموعه آموزشی کوچکتر و یک مجموعه اعتبارسنجی. سپس مدل‌ها روی مجموعه آموزشی کوچکتر آموزش داده شده و عملکردشان روی مجموعه اعتبارسنجی سنجیده می‌شود. این روش کمی زمان‌بر است، اما ساده و مؤثر عمل می‌کند.

یک روش بسیار مناسب دیگر، در کتابخانه‌ی Scikit-Learn بهره‌گیری از اعتبارسنجی متقاطع k-تکه‌ای (*k-fold cross-validation*) است—داده‌ها به k بخش تقسیم می‌شوند و مدل k بار آموزش و آزمایش می‌شود؛ هر بار یک بخش آزمون و بقیه آموزش؛ در این روش، مجموعه آموزشی به‌طور تصادفی به ۱۰ زیرمجموعه غیرهمپوشان به نام *fold* تقسیم می‌شود. سپس مدل درخت تصمیم ده بار آموزش داده و ارزیابی می‌شود، به طوری که در هر بار یک fold برای ارزیابی و ۹ تا fold باقی‌مانده برای آموزش استفاده می‌شوند. خروجی، آرایه‌ای شامل ۱۰ امتیاز ارزیابی خواهد بود:

```
from sklearn.model_selection import cross_val_score

tree_rmses = -cross_val_score(tree_reg, housing, housing_labels,
scoring="neg_root_mean_squared_error", cv=10)
```

هشدار

ویژگی‌های اعتبارسنجی متقاطع در Scikit-Learn به تابعی نیاز دارند که هرچه مقدار آن بیشتر باشد، عملکرد بهتر باشد، نه تابعی که هرچه کمتر باشد بهتر است. از آنجایی که RMSE یک تابع هزینه است (مقادیر کمتر بهترند)، امتیازدهی پیش‌فرض آن به صورت منفی خواهد بود. بنابراین برای به دست آوردن مقدار واقعی RMSE باید علامت خروجی را تغییر دهید.

© 2025 Mohsen Zare. All rights reserved.

بیایید به نتایج نگاهی بیندازیم:

```
>>> pd.Series(tree_rmses).describe()
count        10.000000
mean         66868.027288
std          2060.966425
min          63649.536493
25%          65338.078316
50%          66801.953094
75%          68229.934454
max          70094.778246
dtype: float64
```

در مقایسه با قبل، درخت تصمیم اکنون عملکرد چندان بهتری نسبت به مدل رگرسیون خطی ندارد و تقریباً همان ضعف را نشان می‌دهد. اعتبارسنجی متقاطع نه‌تنها به شما تخمینی از عملکرد واقعی مدل می‌دهد، بلکه امکان محاسبه میزان اطمینان این تخمین، یعنی انحراف معیار آن، را نیز فراهم می‌کند. برای مثال، درخت تصمیم RMSE حدود ۶۶,۸۶۸ با انحراف معیار تقریبی ۲,۰۶۱ دارد. اگر فقط از یک مجموعه اعتبارسنجی استفاده می‌کردید، این اطلاعات قابل دسترسی نبود. البته باید توجه داشت که اعتبارسنجی متقاطع نیازمند آموزش چندباره مدل است و همیشه عملی نیست.

اگر همین معیار را برای مدل رگرسیون خطی محاسبه کنیم، میانگین RMSE تقریباً ۶۹,۸۵۸ و انحراف معیار حدود ۴,۱۸۲ خواهد بود. بنابراین درخت تصمیم تنها کمی بهتر از مدل خطی عمل می‌کند، اما این تفاوت اندک به دلیل بیش‌برازش شدید مدل است. شواهد بیش‌برازش این است که خطای آموزش بسیار کم (در واقع نزدیک صفر) است، در حالی که خطای اعتبارسنجی همچنان بالا باقی می‌ماند.

بیایید آخرین مدل را امتحان کنیم: RandomForestRegressor. همانطور که در فصل ۷ خواهید دید، جنگل‌های تصادفی با آموزش بسیاری از درخت‌های تصمیم‌گیری روی زیرمجموعه‌های تصادفی از ویژگی‌ها و سپس میانگین‌گیری از پیش‌بینی‌های آنها کار می‌کنند. چنین مدل‌هایی که از بسیاری از مدل‌های دیگر تشکیل شده‌اند، «مدل‌های ترکیبی (*ensembles*)»—مجموعه‌ای از مدل‌ها که با هم ترکیب می‌شوند و خروجی حاصل ترکیب این چند مدل است—نامیده می‌شوند: آنها قادر به افزایش عملکرد مدل اصلی (در این مورد، درخت‌های تصمیم‌گیری) هستند. کد تقریباً مشابه کد قبلی است:

```
from sklearn.ensemble import RandomForestRegressor
forest_reg = make_pipeline(preprocessing,
```

© 2025 Mohsen Zare. All rights reserved.

```
                    RandomForestRegressor(random_state=42))
forest_rmses = -cross_val_score(forest_reg, housing, housing_labels,
                        scoring="neg_root_mean_squared_error", cv=10)
```

بریم سراغ امتیازات:

```
>>> pd.Series(forest_rmses).describe()
count       10.000000
mean     47019.561281
std       1033.957120
min      45458.112527
25%      46464.031184
50%      46967.596354
75%      47325.694987
max      49243.765795
dtype: float64
```

خب! این نتیجه بسیار بهتر است: جنگل‌های تصادفی واقعاً عملکرد امیدوارکننده‌ای نشان می‌دهند. با این حال، اگر یک جنگل تصادفی آموزش دهید و RMSE را روی مجموعه آموزشی محاسبه کنید، تقریباً ۱۷,۴۷۴ خواهید دید؛ این مقدار بسیار پایین است و نشان می‌دهد که مدل هنوز بیش‌برازش زیادی دارد. برای کاهش بیش‌برازش، می‌توانید مدل را ساده‌تر کنید، محدودیت‌هایی روی آن اعمال کنید (یعنی منظم‌سازی کنید) یا داده‌های آموزشی بیشتری جمع‌آوری کنید. با این حال، پیش از تمرکز بیشتر روی جنگل‌های تصادفی، بهتر است چند مدل دیگر از دسته‌های مختلف الگوریتم‌های یادگیری ماشین—مانند چندین ماشین بردار پشتیبان با کرنل‌های مختلف و احتمالاً یک شبکه عصبی—را امتحان کنید، بدون آنکه زمان زیادی روی تنظیم هایپرپارامترها بگذارید. هدف این است که دو تا پنج مدل امیدوارکننده را برای فهرست کوتاه خود انتخاب کنید.

© 2025 Mohsen Zare. All rights reserved.

مدل خود را دقیق تنظیم کنید

فرض کنید اکنون یک فهرست از مدل‌های امیدوارکننده دارید. گام بعدی، تنظیم دقیق (fine-tune) این مدل‌ها است تا عملکردشان بهینه شود. در ادامه، به چند روش برای انجام این کار می‌پردازیم.

© 2025 Mohsen Zare. All rights reserved.

جستجوی شبکه‌ای (Grid Search)

یکی از راه‌ها این است که به‌صورت دستی مقادیر مختلف هایپرپارامترها را امتحان کنید تا ترکیب مناسبی پیدا شود. اما این روش بسیار زمان‌بر و خسته‌کننده است و امکان بررسی تعداد زیادی ترکیب را فراهم نمی‌کند.

برای ساده‌تر کردن کار، می‌توان از کلاس GridSearchCV در Scikit-Learn استفاده کرد. کافی است مشخص کنید که کدام هایپرپارامترها و چه مقادیری باید بررسی شوند. سپس این کلاس به کمک اعتبارسنجی متقاطع، همه‌ی ترکیب‌های ممکن از مقادیر هایپرپارامترها را ارزیابی کرده و بهترین گزینه را انتخاب می‌کند.

به عنوان نمونه، کد زیر بهترین ترکیب هایپرپارامترها را برای RandomForestRegressor جستجو می‌کند:

```
from sklearn.model_selection import GridSearchCV

full_pipeline = Pipeline([
  ("preprocessing", preprocessing),
  ("random_forest", RandomForestRegressor(random_state=42)),
])
param_grid = [
  {'preprocessing__geo__n_clusters': [5, 8, 10],
  'random_forest__max_features': [4, 6, 8]},
  {'preprocessing__geo__n_clusters': [10, 15],
  'random_forest__max_features': [6, 8, 10]},
]
grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')
grid_search.fit(housing, housing_labels)
```

نکته مهم این است که در Scikit-Learn می‌توانید به هر هایپرپارامتر مربوط به هر تخمین‌گر درون یک pipeline دسترسی داشته باشید، حتی اگر این تخمین‌گر در لایه‌های عمیق‌تر، مثل چندین pipeline تو در تو یا ColumnTransformerها قرار گرفته باشد.

© 2025 Mohsen Zare. All rights reserved.

برای نمونه، زمانی که Scikit-Learn با رشته‌ی "preprocessing__geo__n_clusters" مواجه می‌شود:

۱. ابتدا رشته را در محل دو زیرخط (__) جدا می‌کند.

۲. سپس به دنبال یک تخمین‌گر با نام "preprocessing" در pipeline می‌گردد. این بخش همان ColumnTransformer است که برای پیش‌پردازش استفاده می‌کنیم.

۳. در مرحله بعد، داخل این ColumnTransformer به دنبال یک ترنسفورمر با نام "geo" می‌گردد. این ترنسفورمر همان ClusterSimilarity است که روی ویژگی‌های latitude و longitude اعمال شده است.

۴. نهایتاً، هایپرپارامتر n_clusters از این ترنسفورمر شناسایی می‌شود.

به همین شکل، عبارت "random_forest__max_features" به هایپرپارامتر max_features از تخمین‌گری با نام "random_forest" اشاره می‌کند. این همان RandomForestRegressor است که در فصل ۷، جزئیات هایپرپارامتر max_features توضیح داده خواهد شد.

نکته

بسته‌بندی مراحل پیش‌پردازش در یک Pipeline از Scikit-Learn این امکان را فراهم می‌کند که هایپرپارامترهای پیش‌پردازش به همراه هایپرپارامترهای مدل به صورت یکپارچه تنظیم شوند. این موضوع اهمیت زیادی دارد زیرا این دو دسته هایپرپارامتر معمولاً با یکدیگر تعامل دارند. برای نمونه، ممکن است افزایش مقدار n_clusters نیازمند افزایش مقدار max_features در مدل باشد. علاوه بر این، اگر فرآیند برازش (fit) ترانسفورمرهای موجود در Pipeline از نظر محاسباتی هزینه‌بر باشد، می‌توان از قابلیت حافظه Pipeline استفاده کرد. برای این کار، مسیر یک دایرکتوری ذخیره‌سازی به پارامتر memory داده می‌شود. در نتیجه، هنگام اولین برازش Pipeline، ترانسفورمرهای برازش‌شده در آن دایرکتوری ذخیره می‌شوند. اگر دوباره Pipeline با همان مقادیر هایپرپارامتر اجرا شود، Scikit-Learn به‌جای برازش مجدد، ترانسفورمرهای ذخیره‌شده را بارگذاری می‌کند.

© 2025 Mohsen Zare. All rights reserved.

در این مثال، param_grid شامل دو دیکشنری است. بنابراین GridSearchCV ابتدا تمام ترکیب‌های ممکن از مقادیر هایپرپارامترها در دیکشنری اول را بررسی می‌کند:

- برای دیکشنری اول: تعداد ترکیب‌ها برابر است با ۹ = ۳ × ۳.
- برای دیکشنری دوم: تعداد ترکیب‌ها برابر است با ۶ = ۳ × ۲.

در نتیجه، کل فرآیند جستجوی شبکه‌ای ۱۵ ترکیب مختلف از مقادیر هایپرپارامترها را ارزیابی خواهد کرد. از آنجا که در اینجا از اعتبارسنجی متقاطع سه-تکه‌ای (3-fold cross-validation) استفاده شده است، هر ترکیب باید ۳ بار آموزش داده شود (هر بار با یک بخش متفاوت از داده‌ها برای اعتبارسنجی). بنابراین در مجموع تعداد دفعات آموزش مدل برابر است با: ۴۵ = ۳ × ۱۵ بار.

این فرآیند ممکن است زمان‌بر باشد، اما در پایان می‌توان بهترین ترکیب مقادیر هایپرپارامتر را استخراج کرد:

```
>>> grid_search.best_params_
{'preprocessing__geo__n_clusters': 15, 'random_forest__max_features': 6}
```

در این آزمایش، بهترین نتیجه مربوط به مدلی است که در آن مقدار n_clusters برابر با ۱۵ و مقدار max_features برابر با ۸ تنظیم شده است.

> **نکته**
>
> از آنجا که مقدار ۱۵ بالاترین مقداری بوده که برای n_clusters آزمایش شده است، منطقی است جستجو را با مقادیر بیشتر نیز تکرار کنید؛ زیرا ممکن است با افزایش تعداد خوشه‌ها، عملکرد مدل همچنان بهبود پیدا کند.

می‌توانید با استفاده از grid_search.best_estimator_ به بهترین مدل دسترسی پیدا کنید. اگر GridSearchCV با پارامتر refit=True (که مقدار پیش‌فرض است) ساخته شده باشد، پس از یافتن بهترین تخمین‌گر توسط اعتبارسنجی متقاطع، آن مدل دوباره روی کل مجموعه‌ی آموزشی آموزش داده

© 2025 Mohsen Zare. All rights reserved.

می‌شود. این کار معمولاً مفید است، زیرا استفاده از کل داده‌های آموزشی می‌تواند عملکرد مدل را بهبود بخشد.

می‌توانید با استفاده از grid_search.cv_results_ به امتیازهای ارزیابی دسترسی داشته باشید. این آبجکت یک دیکشنری است، اما اگر آن را در قالب یک DataFrame قرار دهید، لیستی مرتب از تمام امتیازها آزمون برای هر ترکیب از هایپرپارامترها و برای هر تقسیم‌بندی اعتبارسنجی متقاطع دریافت خواهید کرد. همچنین، میانگین امتیاز آزمون در تمام تقسیم‌بندی‌ها نیز در دسترس خواهد بود:

```
>>> cv_res = pd.DataFrame(grid_search.cv_results_)
>>> cv_res.sort_values(by="mean_test_score", ascending=False, inplace=True)
>>> [...] # change column names to fit on this page, and show rmse = -score
>>> cv_res.head() # note: the 1st column is the row ID
n_clusters max_features split0 split1 split2 mean_test_rmse
12         15          6      43460  43919  44748       44042
13         15          8      44132  44075  45010       44406
14         15         10      44374  44286  45316       44659
7          10          6      44683  44655  45657       44999
9          10          6      44683  44655  45657       44999
```

میانگین RMSE آزمون برای بهترین مدل ۴۴,۰۴۲ است که نسبت به مقدار قبلی ۴۷,۰۱۹ (با هایپرپارامترهای پیش‌فرض) بهبود قابل توجهی دارد. تبریک می‌گوییم، شما توانستید مدل خود را با موفقیت بهینه و تنظیم دقیق کنید.

© 2025 Mohsen Zare. All rights reserved.

جستجوی تصادفی

رویکرد جستجوی شبکه‌ای برای بررسی تعداد محدودی از ترکیب‌های هایپرپارامتر مناسب است، همانند مثال قبلی. با این حال، وقتی فضای جستجو بزرگ باشد، RandomizedSearchCV معمولاً گزینه بهتری است. این روش مشابه GridSearchCV عمل می‌کند، اما به جای آزمون همه ترکیب‌های ممکن، تعداد مشخصی از ترکیب‌ها را به طور تصادفی انتخاب و ارزیابی می‌کند. مزایای این رویکرد عبارت‌اند از:

- برای هایپرپارامترهای پیوسته یا با تعداد زیادی مقدار ممکن، جستجوی تصادفی می‌تواند هزاران مقدار مختلف را بررسی کند، در حالی که جستجوی شبکه‌ای تنها تعداد محدودی از مقادیر تعیین‌شده را امتحان می‌کند.
- اگر برخی هایپرپارامترها تأثیر کمی روی عملکرد مدل داشته باشند، اضافه کردن آنها به جستجوی شبکه‌ای زمان زیادی می‌گیرد، اما جستجوی تصادفی بدون تأثیر قابل توجهی این پارامترها را بررسی می‌کند.
- برای فضای بزرگ با چندین هایپرپارامتر (مثلاً ۶ پارامتر هر کدام با ۱۰ مقدار ممکن)، جستجوی شبکه‌ای نیازمند یک میلیون آموزش مدل است، در حالی که جستجوی تصادفی می‌تواند با هر تعداد تکرار دلخواه اجرا شود.

برای هر هایپرپارامتر باید یک لیست از مقادیر ممکن یا یک توزیع احتمالی مشخص کنید تا RandomizedSearchCV بتواند از آن نمونه‌گیری کند:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {'preprocessing__geo__n_clusters': randint(low=3, high=50),
        'random_forest__max_features': randint(low=2, high=20)}

rnd_search = RandomizedSearchCV(
  full_pipeline, param_distributions=param_distribs, n_iter=10, cv=3,
  scoring='neg_root_mean_squared_error', random_state=42)
```

© 2025 Mohsen Zare. All rights reserved.

```
rnd_search.fit(housing, housing_labels)
```

Scikit-Learn کلاس‌های HalvingRandomSearchCV و HalvingGridSearchCV را ارائه می‌دهد که هدف آن‌ها استفاده بهینه‌تر از منابع محاسباتی است؛ هم برای آموزش سریع‌تر و هم برای بررسی فضای بزرگ‌تری از هایپرپارامترها. عملکرد آن‌ها به این شکل است: ابتدا تعداد زیادی ترکیب هایپرپارامتر (آنها را "کاندیداها" می‌نامند) با روش شبکه‌ای یا تصادفی تولید می‌شوند و روی هر کدام مدلی آموزش داده می‌شود و با اعتبارسنجی متقاطع ارزیابی می‌شوند. اما در این مرحله، آموزش با "منابع محدود" انجام می‌شود، مثلاً فقط با بخشی از داده‌ها یا با تعداد تکرار کمتر. پس از ارزیابی، تنها بهترین کاندیداها به دور بعد می‌روند و در هر دور منابع بیشتری به آن‌ها اختصاص می‌یابد. این فرایند چند دور ادامه پیدا می‌کند تا در نهایت بهترین کاندیداها با استفاده از منابع کامل ارزیابی شوند. این روش می‌تواند در زمان تنظیم هایپرپارامترها صرفه‌جویی قابل توجهی ایجاد کند.

© 2025 Mohsen Zare. All rights reserved.

روش‌های تجمیعی

یکی دیگر از راه‌های بهبود مدل‌ها، استفاده از ترکیب مدل‌هایی است که عملکرد خوبی دارند. این رویکرد که به آن «روش‌های تجمیعی Ensemble Methods»—ترکیب چند مدل برای بهبود عملکرد؛ هدف اصلی: کاهش واریانس، کاهش بایاس، و افزایش دقت مدل است—گفته می‌شود، اغلب بهتر از هر مدل منفرد عمل می‌کند؛ درست مانند جنگل‌های تصادفی که نسبت به درخت‌های تصمیم‌گیری انفرادی که آن‌ها را تشکیل می‌دهند، عملکرد بهتری دارند، به ویژه زمانی که مدل‌های منفرد خطاهای متفاوتی ایجاد کنند. به عنوان مثال، می‌توانید یک مدل k-نزدیک‌ترین همسایه‌ها (k-nearest neighbors) آموزش داده و تنظیم دقیق کنید، سپس یک مدل گروهی بسازید که پیش‌بینی آن، میانگین خروجی جنگل تصادفی و پیش‌بینی مدل kNN باشد. جزئیات بیشتر این روش‌ها در فصل ۷ ارائه خواهد شد.

© 2025 Mohsen Zare. All rights reserved.

تحلیل بهترین مدل‌ها و خطاهای آنها

بررسی دقیق بهترین مدل‌ها اغلب بینش‌های ارزشمندی درباره مسئله فراهم می‌کند. برای نمونه، مدل RandomForestRegressor می‌تواند نشان دهد که هر ویژگی تا چه حد در پیش‌بینی دقیق نقش دارد:

```
>>> final_model = rnd_search.best_estimator_ # includes preprocessing
>>> feature_importances = final_model["random_forest"].feature_importances_
>>> feature_importances.round(2)
array([0.07, 0.05, 0.05, 0.01, 0.01, 0.01, 0.01, 0.19, [...], 0.01])
```

می‌توانیم امتیازهای اهمیت ویژگی‌ها را به ترتیب نزولی مرتب کرده و آنها را همراه با نام ویژگی‌های متناظر نمایش دهیم:

```
>>> sorted(zip(feature_importances,
…         final_model["preprocessing"].get_feature_names_out()),
…         reverse=True)
…
[(0.18694559869103852, 'log__median_income'),
(0.0748194905715524, 'cat__ocean_proximity_INLAND'),
(0.06926417748515576, 'bedrooms__ratio'),
(0.05446998753775219, 'rooms_per_house__ratio'),
(0.05262301809680712, 'people_per_house__ratio'),
(0.03819415873915732, 'geo__Cluster 0 similarity'),
[...]
(0.00015061247730531558, 'cat__ocean_proximity_NEAR BAY'),
(7.301686597099842e-05, 'cat__ocean_proximity_ISLAND')]
```

با استفاده از این داده‌ها، می‌توانید تصمیم بگیرید که برخی ویژگی‌های کم‌اهمیت را حذف کنید. به‌عنوان مثال، به نظر می‌رسد تنها دسته ocean_proximity واقعاً تأثیرگذار است، بنابراین می‌توانید سایر ویژگی‌ها را از مدل حذف کنید.

© 2025 Mohsen Zare. All rights reserved.

نکته

ترانسفورمر SelectFromModel در sklearn.feature_selection می‌تواند ویژگی‌های کم‌اهمیت را به صورت خودکار حذف کند. هنگام برازش، این ترانسفورمر یک مدل (معمولاً جنگل تصادفی) آموزش می‌دهد، به اهمیت ویژگی‌ها (feature_importances_) نگاه می‌کند و مفیدترین ویژگی‌ها را انتخاب می‌کند. سپس با فراخوانی transform()، سایر ویژگی‌ها از داده‌ها حذف می‌شوند.

همچنین باید به خطاهای مشخصی که مدل ایجاد می‌کند توجه کنید و تلاش کنید دلیل آنها را بفهمید و راه‌حل مناسب بیابید؛ برای مثال، می‌توان ویژگی‌های جدید اضافه کرد، ویژگی‌های کم‌فایده را حذف نمود یا داده‌های پرت و نامعتبر را پاکسازی کرد.

اکنون زمان مناسبی است تا مطمئن شوید مدل شما نه تنها به‌طور متوسط، بلکه در تمام زیرگروه‌های مناطق—روستایی و شهری، ثروتمند و فقیر، شمالی و جنوبی، اقلیت و غیر اقلیت و غیره—به خوبی عمل می‌کند. ایجاد زیرمجموعه‌هایی از مجموعه اعتبارسنجی برای هر دسته کمی زمان‌بر است، اما اهمیت زیادی دارد: اگر مدل در کل یک دسته از مناطق عملکرد ضعیفی داشته باشد، تا زمانی که مشکل حل نشود نباید از آن استفاده کرد یا حداقل برای پیش‌بینی آن دسته به کار گرفته شود، زیرا احتمال ضرر آن از فایده بیشتر است.

© 2025 Mohsen Zare. All rights reserved.

سیستم خود را روی مجموعه آزمون ارزیابی کنید

پس از انجام اصلاحات لازم روی مدل‌ها، اکنون سیستمی دارید که عملکرد قابل قبولی ارائه می‌دهد و آماده ارزیابی روی مجموعه آزمون است. روند کار ساده است: داده‌ها و برچسب‌های مجموعه آزمون را آماده کنید، از مدل نهایی final_model خود برای پیش‌پردازش داده‌ها و تولید پیش‌بینی‌ها استفاده کنید، و سپس نتایج را ارزیابی کنید:

```
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

final_predictions = final_model.predict(X_test)

final_rmse = mean_squared_error(y_test, final_predictions, squared=False)
print(final_rmse) # prints 41424.40026462184
```

گاهی یک تخمین نقطه‌ای از خطای تعمیم کافی نیست، مخصوصاً اگر تنها کمی بهتر از مدل فعلی باشد (مثلاً ۰/۱ %). در این شرایط، ممکن است بخواهید دامنه اطمینان این تخمین را نیز بدانید. با استفاده از تابع scipy.stats.t.interval می‌توانید یک بازه اطمینان ۹۵% برای خطای تعمیم محاسبه کنید. در مثال ما، این بازه تقریباً بین ۳۹,۲۷۵ تا ۴۳,۴۶۷ قرار دارد و تخمین نقطه‌ای قبلی یعنی ۴۱,۴۲۴ تقریباً در وسط آن واقع شده است:

```
from scipy import stats
>>> confidence = 0.95
>>> squared_errors = (final_predictions - y_test) ** 2
>>> np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
…                         loc=squared_errors.mean(),
…                         scale=stats.sem(squared_errors)))
…
array([39275.40861216, 43467.27680583])
```

وقتی تعداد زیادی تغییر روی هایپرپارامترها انجام داده باشید، معمولاً عملکرد واقعی کمی پایین‌تر از چیزی است که با اعتبارسنجی متقاطع مشاهده کرده‌اید. دلیلش این است که مدل نهایی طوری بهینه شده که روی داده‌های اعتبارسنجی خوب عمل کند و ممکن است روی داده‌های جدید عملکرد ضعیف‌تری داشته باشد. در مثال ما این مشکل رخ نداده، چون RMSE روی مجموعه آزمون حتی کمتر

© 2025 Mohsen Zare. All rights reserved.

از RMSE اعتبارسنجی است. با این حال، وقتی چنین وضعیتی پیش می‌آید، نباید وسوسه شوید که پارامترها را برای بهتر نشان دادن اعداد روی آزمون تغییر دهید؛ این تغییرات به‌ندرت روی داده‌های جدید تعمیم می‌یابند.

اکنون مرحله پیش از راه‌اندازی پروژه فرا رسیده است: باید راه‌حل خود را ارائه کنید، شامل آنچه یاد گرفته‌اید، روش‌هایی که مؤثر یا غیرمؤثر بوده‌اند، فرضیات اعمال‌شده و محدودیت‌های سیستم. همچنین همه موارد را مستندسازی کرده و ارائه‌هایی با نمودارها و جملات ساده و به‌یادماندنی آماده کنید، مانند «میانگین درآمد، عامل اصلی پیش‌بینی قیمت مسکن است». در مثال مسکن کالیفرنیا، عملکرد نهایی سیستم بهتر از تخمین کارشناسان است که اغلب تا ۳۰% خطا داشتند، اما راه‌اندازی آن هنوز ارزشمند است، به‌ویژه اگر باعث شود کارشناسان زمان بیشتری برای انجام کارهای مهم‌تر و خلاقانه‌تر داشته باشند.

© 2025 Mohsen Zare. All rights reserved.

راه‌اندازی، نظارت و نگهداری سیستم

تبریک! شما مجوز راه‌اندازی سیستم را دریافت کرده‌اید. اکنون باید راه‌حل خود را برای محیط تولید آماده کنید، شامل اصلاح کد، نوشتن مستندات و آزمایش‌ها و سایر مراحل لازم. پس از آماده‌سازی، می‌توانید مدل را در محیط تولید مستقر (deploy) کنید. ساده‌ترین روش، ذخیره‌سازی بهترین مدل آموزش‌دیده، انتقال فایل به محیط تولید و بارگذاری آن است. برای ذخیره مدل می‌توانید از کتابخانه joblib به شکل زیر استفاده کنید:

```
import joblib

joblib.dump(final_model, "my_california_housing_model.pkl")
```

> **نکته**
>
> معمولاً توصیه می‌شود که هر مدلی را که آزمایش می‌کنید ذخیره کنید تا بتوانید در هر زمان به آن بازگردید. علاوه بر این، می‌توانید نمرات اعتبارسنجی متقاطع و حتی پیش‌بینی‌های واقعی روی مجموعه اعتبارسنجی را ذخیره کنید. این کار به شما اجازه می‌دهد عملکرد مدل‌ها را به راحتی مقایسه کرده و انواع خطاهایی که هر مدل تولید می‌کند را بررسی کنید.

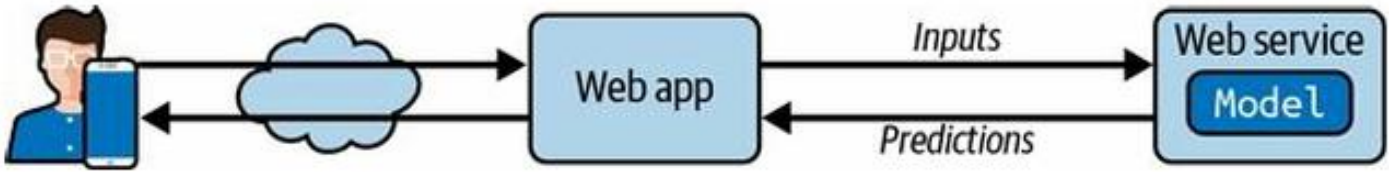
پس از انتقال مدل به محیط تولید، می‌توانید آن را بارگذاری و استفاده کنید. ابتدا باید کلاس‌ها و توابع سفارشی مورد نیاز مدل را وارد کنید تا کد لازم در محیط تولید در دسترس باشد، سپس با استفاده از joblib مدل را بارگذاری کرده و برای پیش‌بینی‌ها از آن استفاده نمایید:

```
import joblib
[...] # import KMeans, BaseEstimator, TransformerMixin, rbf_kernel, etc.

def column_ratio(X): [...]
def ratio_name (function_transformer, feature_names_in): [...]
```

© 2025 Mohsen Zare. All rights reserved.

```
class ClusterSimilarity(BaseEstimator, TransformerMixin): [...]

final_model_reloaded = joblib.load("my_california_housing_model.pkl")

new_data = [...] # some new districts to make predictions for
predictions = final_model_reloaded.predict(new_data)
```

برای نمونه، ممکن است این مدل در یک وب‌سایت کاربرد داشته باشد: کاربر اطلاعات مربوط به یک منطقه جدید را وارد می‌کند و روی دکمه تخمین قیمت کلیک می‌کند. این کار یک پرس‌وجو (query) حاوی داده‌ها به وب‌سرور ارسال می‌کند، سپس وب‌سرور آن را به برنامه وب شما منتقل می‌کند و در نهایت کد شما تنها متد ()predict مدل را فراخوانی می‌کند. مدل بهتر است هنگام راه‌اندازی سرور بارگذاری شود، نه هر بار که استفاده می‌شود. روش دیگر، قرار دادن مدل در یک سرویس وب اختصاصی است که برنامه وب شما بتواند از طریق یک REST API با آن ارتباط برقرار کند (شکل ۲-۲۰). این روش امکان ارتقاء مدل به نسخه‌های جدید بدون توقف برنامه اصلی را آسان می‌کند، مقیاس‌پذیری را ساده‌تر می‌کند—چرا که می‌توانید چندین سرویس وب راه‌اندازی کرده و درخواست‌ها را بین آنها متعادل کنید—و به برنامه وب اجازه می‌دهد از هر زبان برنامه‌نویسی، نه فقط پایتون، استفاده کند.



شکل ۲-۲۰. تصویری از مدلی که به‌صورت یک سرویس وب مستقر شده و توسط یک برنامه وب فراخوانی می‌شود. در این نمودار، درخواست‌های کاربران از برنامه وب به سرویس وب ارسال می‌شوند، سپس سرویس وب داده‌ها را به مدل منتقل کرده و پیش‌بینی‌ها را برمی‌گرداند. این ساختار امکان ارتقاء آسان مدل، مقیاس‌پذیری و استفاده از زبان‌های برنامه‌نویسی مختلف را فراهم می‌کند.

یکی دیگر از روش‌های رایج، انتشار یا استقرار مدل در فضای ابری است، مانند Vertex AI گوگل (قبلاً با نام‌های Google Cloud AI Platform و Google Cloud ML Engine شناخته می‌شد). کافی است مدل خود را با joblib ذخیره کنید و در Google Cloud Storage (GCS) آپلود نمایید، سپس در Vertex AI یک نسخه جدید از مدل بسازید و آن را به فایل GCS ارجاع دهید. با این کار یک سرویس وب ساده خواهید داشت که مقیاس‌پذیری و تعادل بار را برای شما مدیریت می‌کند. این سرویس درخواست‌های JSON حاوی داده‌های ورودی (مثل مشخصات یک منطقه) دریافت می‌کند و پاسخ‌های

© 2025 Mohsen Zare. All rights reserved.

JSON با پیش‌بینی‌ها برمی‌گرداند. در نتیجه، می‌توانید این سرویس وب را در وب‌سایت یا هر محیط تولیدی دیگری استفاده کنید. همانطور که در فصل ۱۹ خواهید دید، استقرار مدل‌های TensorFlow در Vertex AI تفاوت چندانی با مدل‌های Scikit-Learn ندارد.

اما استقرار یا پیاده‌سازی (مدل) پایان کار نیست. شما باید کدهای مانیتورینگ بنویسید تا عملکرد زنده سیستم را در فواصل منظم بررسی کنید و در صورت کاهش عملکرد، هشدارها فعال شوند. افت عملکرد می‌تواند سریع رخ دهد، مثلاً به‌خاطر خرابی یکی از اجزای زیرساخت، یا به آرامی اتفاق بیفتد و برای مدت طولانی نادیده گرفته شود. این پدیده کاملاً رایج است و به دلیل « فرسودگی مدل یا زَوال مدل» رخ می‌دهد: اگر مدل با داده‌های گذشته آموزش دیده باشد، ممکن است با داده‌های فعلی هماهنگ نباشد.

بنابراین، رصد عملکرد زنده مدل ضروری است. نحوه انجام این کار بستگی به نوع سیستم دارد. در برخی موارد، می‌توان عملکرد مدل را از معیارهای پایین‌دستی (downstream metrics) استنباط کرد. برای مثال، اگر مدل شما بخشی از یک سیستم توصیه‌گر باشد و محصولات پیشنهادی را ارائه دهد، می‌توانید تعداد محصولات فروخته‌شده از میان توصیه‌ها را روزانه پایش کنید. کاهش این عدد نسبت به محصولات غیرپیشنهادی، می‌تواند مدل را به عنوان عامل اصلی نشان دهد. این کاهش ممکن است ناشی از خرابی pipeline داده یا نیاز مدل به آموزش مجدد روی داده‌های جدید باشد—موضوعی که به زودی به آن خواهیم پرداخت.

با این حال، گاهی برای ارزیابی عملکرد مدل به «تحلیل انسانی» نیز نیاز است. به عنوان مثال، فرض کنید مدلی برای طبقه‌بندی تصاویر محصول در یک خط تولید آموزش داده‌اید تا نقص‌ها را شناسایی کند. چگونه می‌توانید هنگام اُفت عملکرد، قبل از ارسال هزاران محصول معیوب به مشتریان، هشدار دریافت کنید؟ یکی از راه‌ها این است که نمونه‌ای از تصاویر طبقه‌بندی‌شده—به‌ویژه مواردی که مدل در آن‌ها مطمئن نبوده است—برای ارزیابان انسانی ارسال شود. این ارزیابان ممکن است متخصص باشند یا افراد غیرمتخصص، مانند کارکنان پلتفرم‌های جمع‌سپاری (crowdsourcing) مانند Amazon Mechanical Turk. در برخی برنامه‌ها، خود کاربران نیز می‌توانند نقش ارزیاب را داشته باشند، مثلاً از طریق نظرسنجی یا کپچاهای(captcha) تغییر کاربری داده شده.

در هر حالت، باید یک «سیستم نظارتی» طراحی کنید که با یا بدون ارزیاب‌های انسانی، عملکرد مدل زنده را بررسی کند و همچنین فرآیندهای لازم برای اقدام در صورت بروز خطا و آماده‌سازی برای آن‌ها را

© 2025 Mohsen Zare. All rights reserved.

تعریف نماید. متأسفانه، این بخش اغلب بسیار زمان‌بر است و گاهی بیش از ساخت و آموزش خود مدل به تلاش نیاز دارد.

اگر داده‌ها پیوسته در حال تغییر باشند، لازم است مجموعه داده‌ها را به‌روزرسانی کرده و مدل خود را به طور مرتب آموزش مجدد دهید. بهترین کار این است که کل فرآیند را تا حد امکان خودکار کنید. به عنوان مثال می‌توانید:

- به طور منظم داده‌های جدید جمع‌آوری و برچسب‌گذاری کنید، مثلاً با کمک ارزیاب‌های انسانی.
- اسکریپتی بنویسید که مدل را آموزش دهد و هایپرپارامترها را به صورت خودکار تنظیم کند؛ این اسکریپت می‌تواند به طور منظم، مثلاً روزانه یا هفتگی، اجرا شود.
- اسکریپتی دیگر ایجاد کنید که هم مدل جدید و هم مدل قبلی را روی مجموعه آزمایشی به‌روزرسانی‌شده ارزیابی کند و در صورت حفظ یا بهبود عملکرد، مدل جدید را در محیط عملیاتی مستقر نماید. اگر عملکرد کاهش یافته باشد، حتماً دلیل آن بررسی شود. این ارزیابی باید شامل زیرمجموعه‌های مختلف داده، مانند مناطق فقیر یا ثروتمند، روستایی یا شهری و غیره نیز باشد.

همچنین باید کیفیت داده‌های ورودی مدل را به‌طور مداوم کنترل کنید. گاهی کاهش عملکرد به دلیل داده‌های بی‌کیفیت رخ می‌دهد، مثلاً حسگری خراب داده‌های تصادفی ارسال کند یا خروجی یک تیم دیگر قدیمی شود. این افت ممکن است تا رسیدن به حد هشدار طول بکشد، اما با پایش مستمر ورودی‌ها می‌توانید زودتر متوجه مشکل شوید. برای مثال:

- افزایش تعداد ورودی‌هایی که یک ویژگی خاص را ندارند.
- میانگین یا انحراف معیار ویژگی‌ها از مقادیر آموزش فاصله گرفته باشد.
- یک ویژگی دسته‌بندی‌شده شامل دسته‌های جدید شود.

در چنین شرایطی، سیستم می‌تواند «هشدار فعال کند» تا پیش از ایجاد مشکل جدی، اقدام اصلاحی انجام شود.

© 2025 Mohsen Zare. All rights reserved.

در نهایت، حتماً نسخه پشتیبان از هر مدل خود تهیه کنید و ابزارها و فرآیندهایی برای بازگشت سریع به مدل قبلی داشته باشید، تا اگر مدل جدید به هر دلیلی دچار مشکل شد، بتوانید سریعاً واکنش نشان دهید. داشتن نسخه پشتیبان همچنین امکان مقایسه مدل‌های جدید و قدیمی را به راحتی فراهم می‌کند. به همان ترتیب، نسخه پشتیبان از هر نسخه مجموعه داده‌ها ضروری است تا در صورت خرابی داده‌های جدید (مثلاً پر شدن از داده‌های پرت)، بتوانید به نسخه قبلی بازگردید. این کار همچنین اجازه می‌دهد هر مدل را روی هر نسخه قدیمی از داده‌ها ارزیابی و مقایسه کنید.

همان‌طور که مشاهده می‌کنید، یادگیری ماشین تنها به آموزش مدل محدود نمی‌شود و زیرساخت‌های گسترده‌ای را شامل می‌شود. فصل ۱۹ در جلد دوم ترجمه کتاب، به برخی جنبه‌های این موضوع می‌پردازد، اما در واقع «عملیات یادگیری ماشین (MLOps یا *ML Operations*)» حوزه‌ای بزرگ است که شایسته یک کتاب جداگانه است. بنابراین اگر اولین پروژه شما برای ساخت و استقرار در محیط عملیاتی زمان و تلاش زیادی می‌طلبد، جای تعجب ندارد. خوشبختانه، وقتی این زیرساخت‌ها برقرار شوند، حرکت از ایده تا تولید به شکل قابل‌توجهی سریع‌تر خواهد شد.

© 2025 Mohsen Zare. All rights reserved.

امتحانش کنید!

امیدواریم این فصل تصویری واضح از یک پروژه یادگیری ماشین به شما ارائه کرده باشد و برخی از ابزارهای کاربردی برای آموزش یک سیستم مؤثر را نشان داده باشد. همان‌طور که مشاهده کردید، بخش عمده‌ای از کار در مرحله آماده‌سازی داده‌ها انجام می‌شود: شامل ساخت ابزارهای نظارتی، راه‌اندازی pipelineهای ارزیابی انسانی و خودکارسازی آموزش منظم مدل. البته الگوریتم‌های یادگیری ماشین اهمیت دارند، اما غالباً مفیدتر است که با فرآیند کلی پروژه آشنا باشید و چند الگوریتم (سه یا چهار مورد) را به خوبی بشناسید، تا اینکه تمام وقت خود را صرف بررسی الگوریتم‌های پیشرفته کنید.

بنابراین، اگر هنوز شروع نکرده‌اید، اکنون زمان مناسبی است که یک لپ‌تاپ بردارید، مجموعه داده‌ای که به آن علاقه دارید انتخاب کنید و سعی کنید کل فرآیند یادگیری ماشین را از صفر تا صد اجرا کنید. یک نقطه شروع عالی، وب‌سایت‌های رقابتی مانند Kaggle است: در اینجا می‌توانید با مجموعه داده‌های آماده، اهداف مشخص و جامعه‌ای برای تبادل تجربه کار کنید. از این مسیر لذت ببرید!

© 2025 Mohsen Zare. All rights reserved.

تمرین‌ها

تمرین‌های زیر بر اساس مجموعه داده‌های مسکن این فصل هستند:

۱. با استفاده از کلاس sklearn.svm.SVR یک رگرسیون ماشین بردار پشتیبان (SVR) آموزش دهید. چند ترکیب مختلف از هایپرپارامترها را امتحان کنید؛ برای مثال:
   - کرنل خطی kernel="linear" همراه با مقادیر متفاوت برای هایپرپارامتر C
   - کرنل RBF kernel="rbf" همراه با مقادیر مختلف برای C و gamma

   توجه داشته باشید که الگوریتم‌های SVM در مقیاس‌های بزرگ به‌خوبی عمل نمی‌کنند. بنابراین، مدل خود را فقط روی ۵۰۰۰ نمونه اول از داده‌های آموزشی آموزش دهید و از اعتبارسنجی متقاطع ۳-بخشی (3-fold cross-validation) استفاده کنید. در غیر این صورت، اجرای مدل بسیار زمان‌بر خواهد بود. در این مرحله لازم نیست نگران معنای دقیق هایپرپارامترها باشید؛ این موضوع را در فصل ۵ بررسی خواهیم کرد. در نهایت، عملکرد بهترین مدل SVR خود را گزارش کنید.

۲. به جای استفاده از GridSearchCV، این بار از RandomizedSearchCV استفاده کنید و نتایج آن را بررسی کنید.

۳. یک SelectFromModel به pipeline آماده‌سازی اضافه کنید تا تنها مهم‌ترین ویژگی‌ها انتخاب شوند.

۴. یک ترانسفورمر سفارشی بسازید که در متد fit() یک KNeighborsRegressor (از sklearn.neighbors) را آموزش دهد و در متد transform() خروجی آن را به‌عنوان ویژگی جدید ارائه کند. سپس این ترانسفورمر را به pipeline پیش‌پردازش اضافه کنید و از طول و عرض جغرافیایی به‌عنوان ورودی استفاده کنید. این ویژگی جدید نشان می‌دهد که قیمت متوسط مسکن در نزدیک‌ترین مناطق چقدر است.

۵. با استفاده از GridSearchCV تنظیمات مختلف مربوط به مرحله‌ی آماده‌سازی داده‌ها را به‌طور خودکار بررسی کنید.

© 2025 Mohsen Zare. All rights reserved.

۶. سعی کنید کلاس StandardScalerClone را دوباره از ابتدا پیاده‌سازی کنید، سپس پشتیبانی از متد inverse_transform() را اضافه کنید: اجرای دستور scaler.inverse_transform(scaler.fit_transform(X)) باید آرایه‌ای بسیار نزدیک به **X** را برگرداند. سپس پشتیبانی از نام ویژگی‌ها را اضافه کنید: اگر ورودی یک DataFrame است، feature_names_in_ را در متد fit() تنظیم کنید. این ویژگی باید یک آرایه NumPy از نام ستون‌ها باشد. در نهایت، متد get_feature_names_out() را پیاده‌سازی کنید: این متد باید یک آرگومان اختیاری input_features=None داشته باشد. در صورت ارسال، متد باید بررسی کند که طول آن با n_features_in_ مطابقت داشته باشد، و اگر تعریف شده باشد، باید با feature_names_in_ مطابقت داشته باشد؛ سپس input_features باید برگردانده شود. اگر input_features برابر با None باشد، در این صورت متد باید یا feature_names_in_ را در صورت تعریف شدن برگرداند یا در غیر این صورت np.array(["x0", "x1", ...]) با طول n_features_in_ برگرداند.

پاسخ‌های مربوط به این تمرین‌ها را می‌توانید در انتهای نوت‌بوک همین فصل، از طریق نشانی *https://homl.info/colab3* مشاهده کنید.

© 2025 Mohsen Zare. All rights reserved.

فصل ۳. طبقه‌بندی

در فصل اول ذکر کردم که رایج‌ترین مسائل‌های یادگیری نظارت‌شده، رگرسیون(پیش‌بینی مقادیر) و طبقه‌بندی(پیش‌بینی کلاس‌ها) هستند. در فصل ۲ یک مسئله رگرسیون را بررسی کردیم، پیش‌بینی قیمت‌های مسکن، با استفاده از الگوریتم‌های مختلفی همچون رگرسیون خطی، درختان تصمیم، و جنگل‌های تصادفی (که در فصول بعدی به طور مفصل‌تر توضیح داده خواهند شد). حالا توجه خود را به سیستم‌های طبقه‌بندی معطوف خواهیم کرد.

© 2025 Mohsen Zare. All rights reserved.

MNIST

در این فصل ما از مجموعه داده MNIST استفاده خواهیم کرد، که مجموعه‌ای از ۷۰,۰۰۰ تصویر کوچک از اعداد دست‌نویس است که توسط دانش‌آموزان دبیرستانی و کارکنان اداره سرشماری ایالات متحده نوشته شده است. هر تصویر با عددی که نمایانگر آن است برچسب‌گذاری شده است. این مجموعه آن‌قدر مورد مطالعه قرار گرفته که اغلب به عنوان «Hello World!» در یادگیری ماشین شناخته می‌شود: هرگاه افراد الگوریتم جدیدی برای طبقه‌بندی ارائه دهند، کنجکاوند که ببینند چگونه بر روی MNIST عمل خواهد کرد، و هر کسی که یادگیری ماشین را می‌آموزد، دیر یا زود با این مجموعه داده روبه‌رو می‌شود.

Scikit-Learn توابع کمکی زیادی برای دانلود مجموعه داده‌های محبوب فراهم کرده است. MNIST یکی از این مجموعه داده‌هاست. کد زیر مجموعه داده MNIST را از سایت *OpenML.org* دریافت می‌کند:

```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', as_frame=False)
```

پکیج sklearn.datasets عمدتاً شامل سه نوع تابع است:

- توابعی که با fetch_* شروع می‌شوند (مانند fetch_openml()): این توابع برای دانلود مجموعه داده‌های واقعی از منابعی مثل اینترنت به کار می‌روند.
- توابعی که با load_* شروع می‌شوند: این‌ها مجموعه داده‌های کوچک و آزمایشی (toy datasets) را که به‌صورت پیش‌فرض همراه با Scikit-Learn ارائه می‌شوند، بارگذاری می‌کنند و نیازی به اتصال اینترنت ندارند.
- توابعی که با make_* شروع می‌شوند: این توابع برای تولید مجموعه داده‌های ساختگی (fake datasets) استفاده می‌شوند که معمولاً برای آزمون یا آموزش الگوریتم‌ها مفید هستند.

© 2025 Mohsen Zare. All rights reserved.

مجموعه داده‌های تولیدشده معمولاً به‌صورت یک تاپِل (X, y) بازگردانده می‌شوند که شامل داده‌های ورودی و اهداف (labels) هستند، هر دو به‌شکل آرایه‌های NumPy. سایر مجموعه داده‌ها به‌صورت آبجکت‌هایی (objects) از نوع sklearn.utils.Bunch بازگردانده می‌شوند که مانند دیکشنری عمل می‌کنند، با این تفاوت که می‌توان به اعضای آن‌ها به‌صورت ویژگی‌ها (attribute) نیز دسترسی داشت. این آبجکت‌ها معمولاً شامل موارد زیر هستند:

"DESCR"

شرح و توضیحی از مجموعه داده

"data"

داده‌های ورودی، معمولاً به‌صورت یک آرایه دو بُعدی از نوع NumPy

"target"

برچسب‌ها، معمولاً به صورت یک آرایه یک بُعدی از نوع NumPy

تابع fetch_openml() کمی خاص است چون به‌طور پیش‌فرض ورودی‌ها را به صورت یک DataFrame از کتابخانه Pandas و برچسب‌ها را به صورت یک Series از Pandas برمی‌گرداند (مگر اینکه مجموعه داده پراکنده باشد). اما مجموعه داده MNIST شامل تصاویر است و DataFrameها برای این کار مناسب نیستند، بنابراین بهتر است مقدار as_frame=False تنظیم شود تا داده‌ها به صورت آرایه‌های NumPy دریافت شوند. بیایید به این آرایه‌ها نگاهی بیندازیم:

```
>>> X, y = mnist.data, mnist.target
>>> X
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
>>> X.shape
```

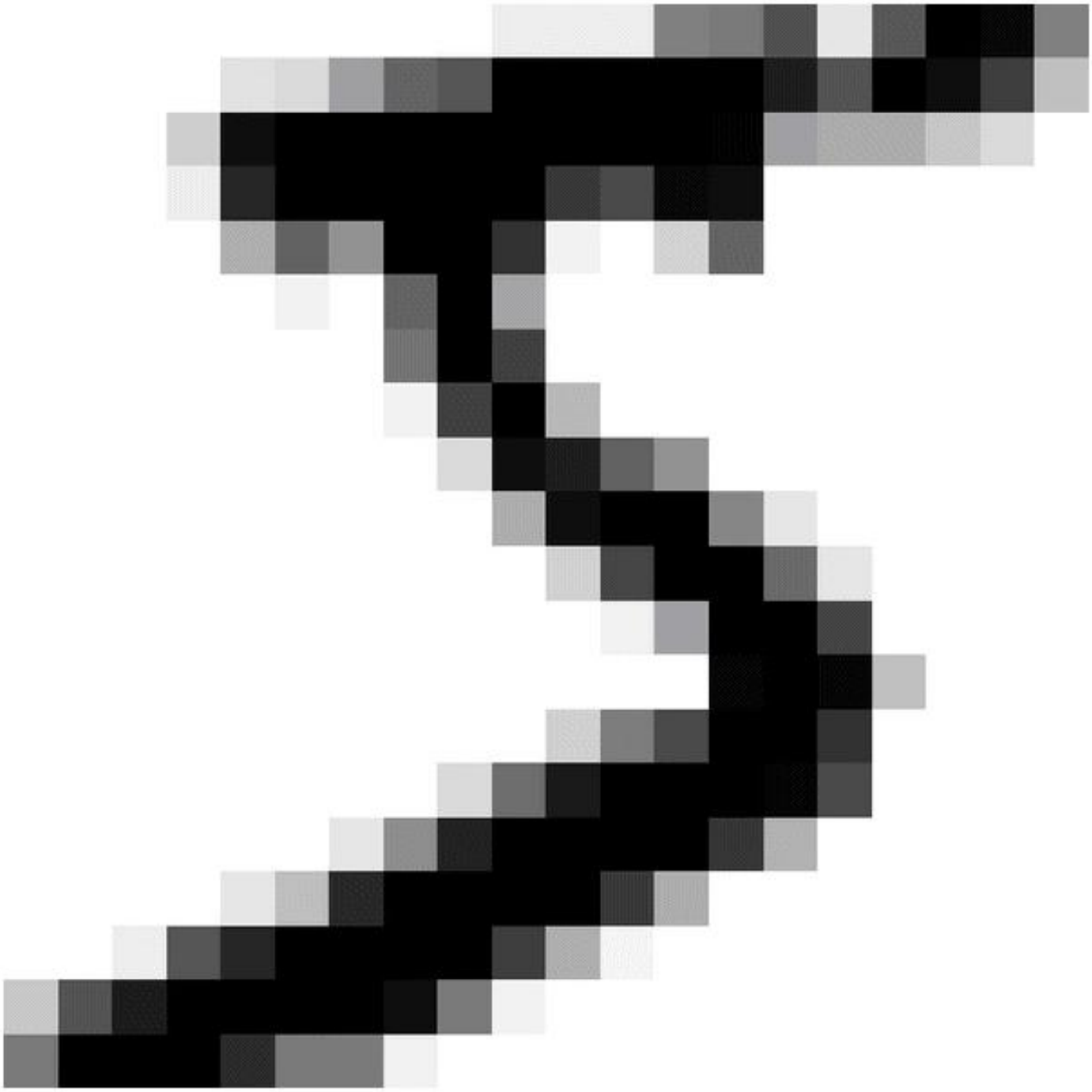
© 2025 Mohsen Zare. All rights reserved.

```
(70000, 784)
>>> y
array(['5', '0', '4', ..., '4', '5', '6'], dtype=object)
>>> y.shape
(70000,)
```

۷۰,۰۰۰ تصویر وجود دارد و هر تصویر دارای ۷۸۴ ویژگی است. دلیل این امر این است که هر تصویر ۲۸×۲۸ پیکسل دارد و هر ویژگی به سادگی شدت یک پیکسل را نشان می‌دهد، از ۰ (سفید) تا ۲۵۵ (سیاه). بیایید نگاهی به یکی از اعداد موجود در مجموعه داده بیندازیم (شکل ۳-۱). کافی است بردار ویژگی یک نمونه را برداریم، آن را به آرایه‌ای به اندازه ۲۸ در ۲۸ تغییر شکل دهیم و سپس با استفاده از تابع imshow() در کتابخانه Matplotlib نمایش دهیم. برای اینکه نقشه رنگی خاکستری داشته باشیم که ۰ سفید و ۲۵۵ سیاه باشد، از cmap="binary" استفاده می‌کنیم:

```
import matplotlib.pyplot as plt

def plot_digit(image_data):
  image = image_data.reshape(28, 28)
  plt.imshow(image, cmap="binary")
  plt.axis("off")
  some_digit = X[0]
plot_digit(some_digit)
plt.show()
```

© 2025 Mohsen Zare. All rights reserved.



شکل ۳-۱. نمونه‌ای از یک تصویر MNIST

این تصویر شبیه عدد ۵ به نظر می‌رسد، و در واقع برچسب آن نیز همین را نشان می‌دهد:

```
>>> y[0]
'5'
```

© 2025 Mohsen Zare. All rights reserved.

برای اینکه دید بهتری از میزان پیچیدگی مسئله‌ی طبقه‌بندی داشته باشید، شکل ۳-۲ چند تصویر دیگر از مجموعه داده MNIST را نشان می‌دهد.

اما صبر کنید! همیشه باید پیش از بررسی دقیق داده‌ها، یک مجموعه‌ی آزمون(test set) ایجاد کرده و آن را کنار بگذارید. مجموعه داده MNIST که توسط تابع fetch_openml() بازگردانده می‌شود، در واقع از پیش به دو بخش تقسیم شده است: مجموعه آموزش شامل ۶۰,۰۰۰ تصویر اول و مجموعه آزمون شامل ۱۰,۰۰۰ تصویر آخر:

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

مجموعه داده‌ی آموزش از قبل به‌صورت تصادفی مرتب(shuffle) شده است، که نکته‌ی مثبتی به حساب می‌آید؛ چراکه این کار تضمین می‌کند تمام زیرمجموعه‌های اعتبارسنجی متقاطع (cross-validation folds) مشابه یکدیگر باشند—و از این طریق، از وقوع حالتی که در آن یک بخش یا fold فاقد برخی رقم‌ها باشد، جلوگیری می‌شود. علاوه بر این، برخی الگوریتم‌های یادگیری به ترتیبِ داده‌های آموزشی حساس هستند و در صورتی که نمونه‌های مشابه به‌صورت متوالی به آن‌ها داده شوند، عملکرد ضعیفی از خود نشان می‌دهند. تصادفی‌سازی داده‌ها(شافِل کردن) باعث می‌شود چنین الگوریتم‌هایی با داده‌هایی متنوع‌تر و بهینه‌تر آموزش ببینند.

© 2025 Mohsen Zare. All rights reserved.

شکل ۳-۲. نمونه‌هایی از ارقام مجموعه داده MNIST

© 2025 Mohsen Zare. All rights reserved.

آموزش یک طبقه‌بند باینری(دودویی)

بیایید فعلاً مسئله را ساده‌تر کنیم و فقط سعی کنیم یک رقم خاص را شناسایی کنیم—برای مثال عدد ۵. این «کاشفِ عدد ۵» نمونه‌ای از یک «دسته‌بند یا طبقه‌بند باینری (*binary classifier*)» خواهد بود؛ مدلی که توانایی تمایز بین تنها دو کلاس را دارد: ۵ و غیر از ۵. ابتدا، بردارهای هدف (target vectors) را برای این مسئله‌ی طبقه‌بندی ایجاد می‌کنیم:

```
y_train_5 = (y_train == '5')  # True for all 5s, False for all other digits
y_test_5 = (y_test == '5')
```

حالا بیایید یک مدل طبقه‌بندی انتخاب کنیم و آن را آموزش دهیم. یکی از انتخاب‌های مناسب برای شروع، استفاده از طبقه‌بند «گرادیانِ نزولیِ تصادفی (SGD: *Stochastic Gradient Descent*)» است، که می‌توان آن را از کلاس SGDClassifier در کتابخانه Scikit-Learn فراخوانی کرد.

این مدلِ طبقه‌بند، توانایی بالایی در پردازش مجموعه داده‌های بسیار بزرگ دارد. یکی از دلایل آن این است که SGD نمونه‌های آموزشی را به‌صورت مستقل و یکی‌یکی پردازش می‌کند، که همین ویژگی آن را برای یادگیری آنلاین نیز بسیار مناسب می‌سازد (که در ادامه با آن بیشتر آشنا می‌شوید). بیایید یک آبجکت از کلاس SGDClassifier بسازیم و آن را روی کل داده‌های آموزشی، آموزش دهیم:

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

اکنون می‌توانیم از این مدل برای تشخیص تصاویر عدد ۵ استفاده کنیم:

```
>>> sgd_clf.predict([some_digit])
array([ True])
```

این مدل پیش‌بینی می‌کند که تصویر موردنظر عدد ۵ را نشان می‌دهد (پاسخ: درست). به نظر می‌رسد در این مورد خاص، پیش‌بینی‌اش صحیح بوده است! حالا بیایید عملکرد این مدل را ارزیابی کنیم.

© 2025 Mohsen Zare. All rights reserved.

معیارهای ارزیابی عملکرد

ارزیابی یک مدل طبقه‌بندی(classification) معمولاً بسیار دشوارتر از ارزیابی یک مدل رگرسیون است، بنابراین بخش زیادی از این فصل را به این موضوع اختصاص خواهیم داد. معیارهای زیادی برای سنجش عملکرد مدل‌های طبقه‌بندی وجود دارد، پس یک فنجان قهوه دیگر برای خودت بریز و آماده باش تا با کلی مفهوم و اصطلاح جدید آشنا شوی!

© 2025 Mohsen Zare. All rights reserved.

سنجش دقت با استفاده از اعتبارسنجی متقاطع

یکی از روش‌های مناسب برای ارزیابی یک مدل، استفاده از اعتبارسنجی متقاطع است، همان‌طور که در فصل ۲ انجام دادید. بیایید از تابع cross_val_score() برای ارزیابی مدل SGDClassifier خود استفاده کنیم، با استفاده از اعتبارسنجی متقاطع k-بخشی (k-fold). به خاطر داشته باشید که اعتبارسنجی k-بخشی به این معناست که مجموعه‌ی آموزش به k تِکه تقسیم می‌شود (در اینجا ۳ تِکه)، سپس مدل k بار آموزش داده می‌شود و هر بار یکی از بخش‌ها برای ارزیابی کنار گذاشته می‌شود (به فصل ۲ نگاه کنید):

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.95035, 0.96035, 0.9604 ])
```

اوه! دقتی بالای ۹۵% (یعنی نسبت پیش‌بینی‌های درست) در تمام بخش‌های اعتبارسنجی متقاطع؟ فوق‌العاده به نظر می‌رسه، نه؟ خب، قبل از اینکه خیلی هیجان‌زده بشوید، بیایید نگاهی به یک مدل کاملاً ساده بیندازیم؛ مدلی که همه تصاویر را صرفاً به پرکاربردترین کلاس نسبت می‌دهد—که در این مثال، کلاس منفی (یعنی عددی غیر از ۵) است:

```
from sklearn.dummy import DummyClassifier

dummy_clf = DummyClassifier()
dummy_clf.fit(X_train, y_train_5)
print(any(dummy_clf.predict(X_train)))  # prints False: no 5s detected
```

می‌توانید حدس بزنید دقت این مدل چقدر خواهد بود؟ بیایید ببینیم:

```
>>> cross_val_score(dummy_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.90965, 0.90965, 0.90965])
```

بله، این مدل دقتی بیش از ۹۰% دارد! دلیلش خیلی ساده است: چون فقط حدود ۱۰% از تصاویر مربوط به عدد ۵ هستند، اگر همیشه حدس بزنید که تصویر مربوط به عدد ۵ «نیست»، در حدود ۹۰% مواقع درست گفته‌اید. از نوسترآداموس هم جلو زده است!

© 2025 Mohsen Zare. All rights reserved.

این موضوع نشان می‌دهد که چرا دقت (accuracy) معمولاً معیار مناسبی برای ارزیابی عملکرد طبقه‌بندها نیست، به‌ویژه وقتی با «مجموعه داده‌های نامتوازن (*skewed datasets*)» سروکار داریم (یعنی زمانی که برخی کلاس‌ها بسیار رایج‌تر و پرتکرارتر از بقیه هستند). روش بسیار بهتری برای ارزیابی عملکرد یک مدل طبقه‌بندی، بررسی «ماتریس آشفتگی یا ماتریس درهم‌ریختگی (CM: *confusion matrix*)» است.

پیاده‌سازی اعتبارسنجی متقاطع

گاهی اوقات نیاز دارید کنترل بیشتری روی فرایند اعتبارسنجی متقاطع داشته باشید، چیزی فراتر از آنچه Scikit-Learn به‌صورت آماده ارائه می‌دهد. در چنین مواردی، می‌توانید خودتان فرآیند cross-validation را پیاده‌سازی کنید. کد زیر تقریباً همان کاری را انجام می‌دهد که تابع cross_val_score() در Scikit-Learn انجام می‌دهد، و همان نوع نتیجه را چاپ می‌کند:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3)  # add shuffle=True if the dataset is
                                       # not already shuffled
for train_index, test_index in skfolds.split(X_train, y_train_5):
  clone_clf = clone(sgd_clf)
  X_train_folds = X_train[train_index]
  y_train_folds = y_train_5[train_index]
  X_test_fold = X_train[test_index]
  y_test_fold = y_train_5[test_index]

  clone_clf.fit(X_train_folds, y_train_folds)
  y_pred = clone_clf.predict(X_test_fold)
  n_correct = sum(y_pred == y_test_fold)
  print(n_correct / len(y_pred))  # prints 0.95035, 0.96035, and 0.9604
```

© 2025 Mohsen Zare. All rights reserved.

کلاس StratifiedKFold نمونه‌برداریِ دسته‌بندی‌شده انجام می‌دهد (همان‌طور که در فصل ۲ توضیح داده شد) تا foldهایی ایجاد کند که نسبتِ نماینده‌گی (representative ratio) از هر کلاس را شامل شوند. در هر تکرار، کد یک کلون(شبیه‌سازی) از دسته‌بند ایجاد می‌کند، آن را روی foldهای آموزشی آموزش می‌دهد، و روی fold آزمایشی پیش‌بینی انجام می‌دهد. سپس تعداد پیش‌بینی‌های صحیح را می‌شمارد و نسبت پیش‌بینی‌های درست را چاپ می‌کند.

© 2025 Mohsen Zare. All rights reserved.

ماتریس درهم‌ریختگی

ایده‌ی کلی ماتریس درهم‌ریختگی این است که تعداد دفعاتی را بشماریم که نمونه‌های کلاس A به اشتباه به عنوان کلاس B طبقه‌بندی شده‌اند، برای تمام جفت‌های A/B. برای مثال، اگر بخواهید بدانید طبقه‌بند چند بار تصاویر عدد ۸ را با تصاویر عدد ۰ اشتباه گرفته، باید به سطر شماره ۸ و ستون شماره ۰ ماتریس درهم‌ریختگی نگاه کنید.

برای محاسبه‌ی ماتریس درهم‌ریختگی، ابتدا باید مجموعه‌ای از پیش‌بینی‌ها داشته باشید تا بتوانید آن‌ها را با مقادیر واقعی مقایسه کنید. می‌توانید روی مجموعه‌ی آزمون پیش‌بینی انجام دهید، اما بهتر است فعلاً آن را دست‌نخورده نگه دارید (به یاد داشته باشید که مجموعه‌ی آزمون را باید فقط در مرحله‌ی نهایی پروژه و زمانی که مدل نهایی آماده‌ی استفاده است، به کار ببرید). در عوض، می‌توانید از تابع cross_val_predict() استفاده کنید:

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

درست مانند تابع cross_val_score()، تابع cross_val_predict() نیز اعتبارسنجی متقاطع k-بخشی را انجام می‌دهد، اما به‌جای بازگرداندن امتیازهای ارزیابی، پیش‌بینی‌هایی را که روی هر زیرمجموعه‌ی آزمون (test fold) انجام شده‌اند، برمی‌گرداند. این بدان معناست که برای هر نمونه در مجموعه‌ی آموزش، یک پیش‌بینی «تمیز» (clean) دریافت می‌کنید (منظور از «تمیز»، «خارج از نمونه» یا out-of-sample است: مدلی که پیش‌بینی می‌کند، داده‌هایی را پیش‌بینی می‌کند که هنگام آموزش هرگز آن‌ها را ندیده است).

الآن آماده‌اید تا با استفاده از تابع confusion_matrix()، ماتریس درهم‌ریختگی را به دست آورید. فقط کافی است کلاس‌های هدف (y_train_5) و کلاس‌های پیش‌بینی‌شده (y_train_pred) را به آن بدهید:

```
>>> from sklearn.metrics import confusion_matrix
>>> cm = confusion_matrix(y_train_5, y_train_pred)
```

© 2025 Mohsen Zare. All rights reserved.

```
>>> cm
array([[53892,   687],
       [ 1891,  3530]])
```

هر سطر در یک ماتریس درهم‌ریختگی نشان‌دهنده‌ی یک «کلاس واقعی (*actual class*)» است، در حالی‌که هر ستون نشان‌دهنده‌ی یک «کلاس پیش‌بینی‌شده (*predicted class*)» است. سطر اول این ماتریس مربوط به تصاویر غیر از عدد ۵ است («کلاس منفی(*negative class*)»): تعداد ۵۳,۸۹۲ تصویر به‌درستی به‌عنوان «غیرِ ۵» طبقه‌بندی شده‌اند (که به آن‌ها «منفی‌های واقعی (*true negatives*)» گفته می‌شود)، در حالی‌که ۶۸۷ تصویر به‌اشتباه به‌عنوان عدد ۵ طبقه‌بندی شده‌اند («مثبت کاذب (*false positives*)»، که به آن‌ها «خطاهای نوع اول (*type I errors*)» نیز گفته می‌شود). سطر دوم مربوط به تصاویر عدد ۵ است («کلاس مثبت (*positive class*)»): تعداد ۱,۸۹۱ تصویر به‌اشتباه به‌عنوان «غیر ۵» طبقه‌بندی شده‌اند («منفی کاذب (*false negatives*)»، که به آن‌ها «خطاهای نوع دوم (*type II errors*)» نیز گفته می‌شود)، در حالی‌که ۳,۵۳۰ تصویر به‌درستی به‌عنوان عدد ۵ طبقه‌بندی شده‌اند («مثبت واقعی (*true positives*)»). یک مدل طبقه‌بندی بی‌نقص فقط شامل مثبت‌های واقعی و منفی‌های واقعی خواهد بود، بنابراین ماتریس درهم‌ریختگی آن فقط روی قطر اصلی (از بالا سمت چپ تا پایین سمت راست) مقادیر غیرصفر خواهد داشت:

```
>>> y_train_perfect_predictions = y_train_5  # pretend we reached perfection
>>> confusion_matrix(y_train_5, y_train_perfect_predictions)
array([[54579,     0],
       [0,     5421]])
```

ماتریس درهم‌ریختگی اطلاعات زیادی به شما می‌دهد، اما گاهی ممکن است ترجیح بدهید از یک معیار خلاصه‌تر استفاده کنید. یکی از معیارهای جالب، دقت پیش‌بینی‌های مثبت است؛ به این معیار، «دقت مثبت‌ها (*precision*)» طبقه‌بند گفته می‌شود (معادله ۳-۱).

معادله ۳-۱. دقت

$$\text{precision} = \frac{TP}{TP + FP}$$

© 2025 Mohsen Zare. All rights reserved.

TP تعداد پیش‌بینی‌های مثبت درست (True Positives) است و FP تعداد پیش‌بینی‌های مثبت نادرست (False Positives) می‌باشد.

یک راه ساده برای دستیابی به دقت کامل این است که یک طبقه‌بند بسازید که همیشه پیش‌بینی منفی انجام دهد، به‌جز یک پیش‌بینی مثبت روی نمونه‌ای که بیشترین اطمینان را نسبت به آن دارد. اگر این یک پیش‌بینی درست باشد، آنگاه دقت دسته‌بند ۱۰۰% خواهد بود (دقت = ۱ تقسیم بر ۱ = ۱۰۰%). بدیهی است که چنین طبقه‌بند چندان مفید نخواهد بود، چون تمام نمونه‌های مثبت دیگر به جز یکی را نادیده می‌گیرد. بنابراین، معمولاً دقت همراه با معیار دیگری به نام « نرخ شناسایی مثبت‌ها (*recall*)» استفاده می‌شود که با نام‌های «حساسیت (*sensitivity*)» یا «نرخ مثبت واقعی (TPR: *True Positive Rate*)» نیز شناخته می‌شود: این معیار نسبت نمونه‌های مثبت‌ای است که به‌درستی توسط طبقه‌بند شناسایی شده‌اند (معادله ۳-۲).

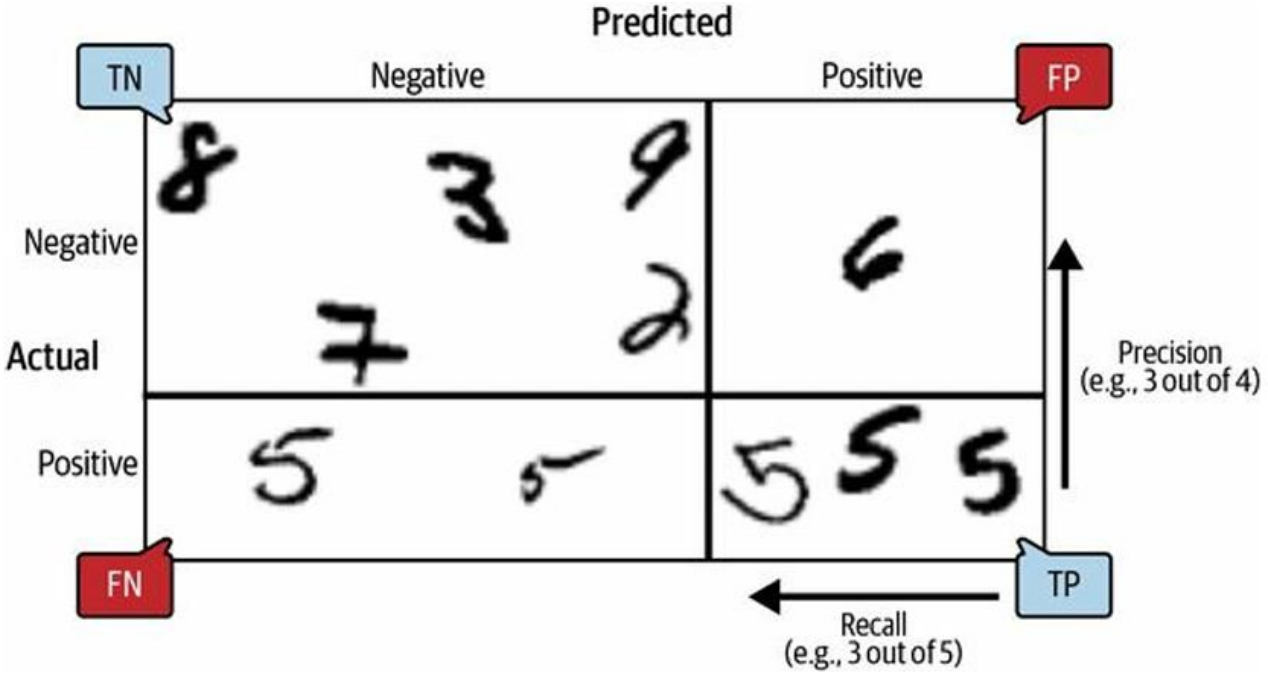
معادله ۳-۲. recall

$$\text{Recall} = \frac{TP}{TP + FN}$$

FN البته، تعداد نمونه‌های مثبت نادرستی است که به اشتباه منفی پیش‌بینی شده‌اند که منفی کاذب (False Negatives) نامیده می‌شوند.

اگر در مورد ماتریس درهم‌ریختگی سردرگم هستید، شکل ۳-۳ می‌تواند کمک‌کننده باشد.

© 2025 Mohsen Zare. All rights reserved.



شکل ۳-۳. یک ماتریس درهم‌ریختگی مصور که نمونه‌هایی از منفی‌های واقعی (بالا سمت چپ)، مثبت‌های کاذب (بالا سمت راست)، منفی‌های کاذب (پایین سمت چپ) و مثبت‌های واقعی (پایین سمت راست) را نشان می‌دهد.

© 2025 Mohsen Zare. All rights reserved.

دقت و recall

کتابخانه Scikit-Learn چندین تابع برای محاسبه معیارهای طبقه‌بندی، از جمله دقت و recall، ارائه می‌دهد:

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred)  # == 3530 / (687 + 3530)
0.8370879772350012
>>> recall_score(y_train_5, y_train_pred)  # == 3530 / (1891 + 3530)
0.6511713705958311
```

حالا «شناساگر عدد ۵» یا (5-detector) ما دیگر آنقدر درخشان به نظر نمی‌رسد به اندازه‌ی وقتی که دقت آن را بررسی کردیم. وقتی ادعا می‌کند یک تصویر عدد ۵ است، تنها در ۸۳٫۷ % مواقع درست حدس می‌زند. علاوه بر این، تنها ۶۵٫۱ % از اعداد ۵ را شناسایی می‌کند.

ترکیب دقت و recall در قالب یک معیار واحد، که «امتیاز F_1 ($F_1 - score$)» نامیده می‌شود، اغلب بسیار کاربردی است؛ به‌ویژه زمانی‌که برای مقایسه دو طبقه‌بند نیاز به یک معیار واحد دارید. امتیاز F_1 «میانگینِ هارمونیک (*harmonic mean*)» دقت و recall است (معادله ۳-۳). برخلاف میانگین معمولی که همه مقادیر را به یک اندازه در نظر می‌گیرد، میانگین هارمونیک به مقادیر پایین وزن بیشتری می‌دهد. در نتیجه، تنها زمانی طبقه‌بند، امتیاز F_1 بالایی می‌گیرد که هم دقت و هم recall بالا باشند.

معادله ۳-۳. امتیاز F_1

$$\mathrm{F}_1 = 2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} = \frac{2\ \mathrm{TP}}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP}}$$

برای محاسبه امتیاز F_1، کافی است تابع f1_score() را فراخوانی کنید:

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
```

© 2025 Mohsen Zare. All rights reserved.

0.7325171197343846

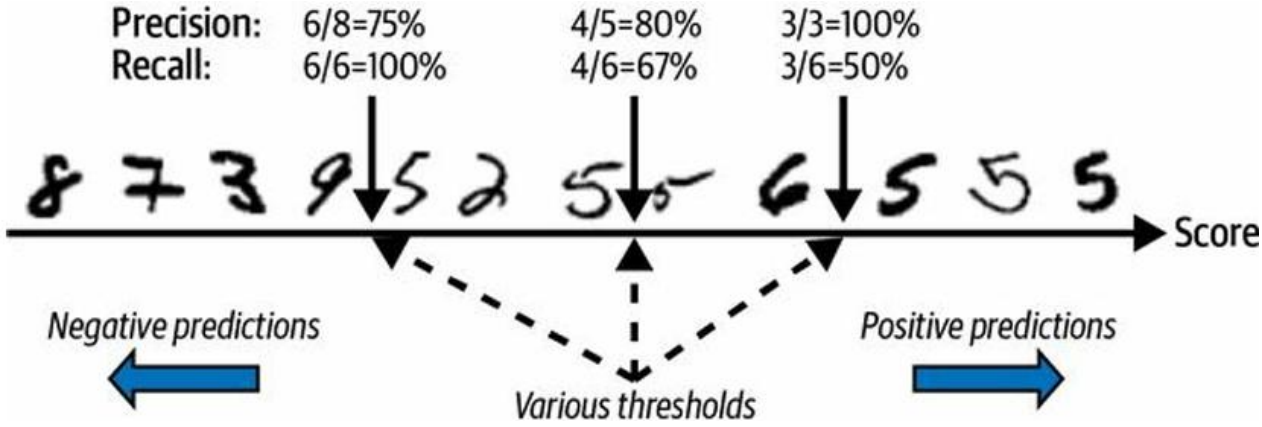
امتیاز F_1 به طبقه‌بندهایی تمایل دارد که دقت و recall مشابهی دارند. اما این همیشه چیزی نیست که شما بخواهید: در برخی موقعیت‌ها بیشتر به دقت اهمیت می‌دهید، و در برخی موقعیت‌های دیگر recall برایتان مهم‌تر است. برای مثال، اگر یک طبقه‌بند برای تشخیص ویدیوهای مناسب برای کودکان آموزش داده باشید، احتمالاً ترجیح می‌دهید طبقه‌بندی داشته باشید که بسیاری از ویدیوهای خوب را رد کند (recall پایین) اما فقط ویدیوهای واقعاً امن را نگه دارد (دقت بالا)، نه اینکه طبقه‌بندی با recall بالا داشته باشید که اجازه دهد چند ویدیوی واقعاً بد در محصول شما نمایش داده شود. (در چنین مواردی حتی ممکن است بخواهید یک فرآیند پردازش داده‌ی انسانی برای بررسی ویدیوهای انتخاب‌شده توسط طبقه‌بند اضافه کنید). از طرف دیگر، فرض کنید یک طبقه‌بند برای شناسایی دزدهای فروشگاه در تصاویر دوربین‌های مداربسته آموزش داده‌اید: در این حالت احتمالاً مشکلی ندارد اگر طبقه‌بند شما فقط ۳۰% دقت داشته باشد، به شرطی که recall آن ۹۹% باشد (درسته، نگهبان‌های امنیتی چند هشدار اشتباه دریافت می‌کنند، اما تقریباً همه دزدها شناسایی خواهند شد).

متأسفانه نمی‌توان هر دو را هم‌زمان داشت: افزایش دقت باعث کاهش recall می‌شود و بالعکس. به این وضعیت، «موازنهٔ دقت-recall (*precision/recall trade-off*)» گفته می‌شود.

© 2025 Mohsen Zare. All rights reserved.

موازنه دقت-recall

برای درک این موازنه، بیایید ببینیم مدل SGDClassifier چگونه در مورد طبقه‌بندی تصمیم‌گیری می‌کند. این مدل برای هر نمونه، یک امتیاز محاسبه می‌کند که بر پایه‌ی یک «تابع تصمیم‌گیری (*decision function*)» است. اگر آن امتیاز از یک آستانهٔ (threshold) مشخص بیشتر باشد، مدل نمونه را به کلاسِ مثبت اختصاص می‌دهد؛ در غیر این صورت، آن را در کلاس منفی قرار می‌دهد. شکل ۴-۳ چند رقم را از سمت چپ (کمترین امتیاز) تا سمت راست (بیشترین امتیاز) نشان می‌دهد. فرض کنید «آستانه‌ی تصمیم‌گیری (*decision threshold*)» در محل پیکان مرکزی (بین دو عدد ۵) قرار دارد: در این صورت، در سمت راست آستانه، ۴ نمونه مثبت واقعی (TP؛ یعنی واقعاً عدد ۵) قرار دارند و تنها ۱ نمونه مثبت کاذب (FP؛ در واقع عدد ۶ است) دیده می‌شود. بنابراین، با این آستانه، دقت برابر با ۸۰% است (۴ مورد صحیح از ۵ مورد پیش‌بینی‌شده). اما از بین ۶ عدد ۵ واقعی، مدل فقط ۴ مورد را تشخیص داده است، پس recall برابر با ۶۷% است (۴ مورد از ۶). اگر آستانه را افزایش دهید (آن را به سمت فلش راست حرکت دهید)، مثبت کاذب (عدد ۶) تبدیل به منفی واقعی می‌شود و در نتیجه دقت افزایش می‌یابد (در این مثال تا ۱۰۰%)، اما یکی از مثبت‌های واقعی تبدیل به منفی کاذب می‌شود و میزانِ recall کاهش می‌یابد (تا ۵۰%). برعکس، کاهش آستانه باعث افزایش recall و کاهش دقت می‌شود.



شکل ۴-۳. موازنهٔ بین دقت و recall: تصاویر بر اساس امتیاز خروجی طبقه‌بند مرتب شده‌اند، و آن‌هایی که امتیازشان بالاتر از آستانه تصمیم انتخاب‌شده است، به‌عنوان نمونه‌ی مثبت در نظر گرفته می‌شوند؛ هرچه آستانه بالاتر باشد، recall کمتر و (به‌طور کلی) دقت بیشتر خواهد بود.

© 2025 Mohsen Zare. All rights reserved.

کتابخانه Scikit-Learn اجازه نمی‌دهد آستانه را به‌صورت مستقیم تنظیم کنید، اما به امتیازهای تصمیم‌گیری(decision scores) که برای پیش‌بینی استفاده می‌کند، دسترسی می‌دهد. به‌جای فراخوانی متد predict()، می‌توانید از متد decision_function() استفاده کنید که برای هر نمونه یک امتیاز بازمی‌گرداند، و سپس با استفاده از هر آستانه‌ای که بخواهید، بر اساس این امتیازها پیش‌بینی انجام دهید:

```
>>> y_scores = sgd_clf.decision_function([some_digit])
>>> y_scores
array([2164.22030239])
>>> threshold = 0
>>> y_some_digit_pred = (y_scores > threshold)
array([ True])
```

کلاس SGDClassifier از آستانه‌ی برابر با ۰ استفاده می‌کند، بنابراین کدی که در بالا دیدید، همان نتیجه‌ای را بازمی‌گرداند که متد predict() برمی‌گرداند (یعنی مقدار True). حالا بیایید آستانه را بالاتر ببریم:

```
>>> threshold = 3000
>>> y_some_digit_pred = (y_scores > threshold)
>>> y_some_digit_pred
array([False])
```

این موضوع تأیید می‌کند که افزایش آستانه باعث کاهش recall می‌شود. تصویر در واقع عدد ۵ را نشان می‌دهد و وقتی آستانه برابر ۰ است، طبقه‌بند آن را تشخیص می‌دهد، اما وقتی آستانه به ۳,۰۰۰ افزایش می‌یابد، آن را از دست می‌دهد.

چطور تصمیم می‌گیرید که کدام آستانه را استفاده کنید؟ اول، از تابع cross_val_predict() استفاده کنید تا امتیازهای تمام نمونه‌ها در مجموعه آموزش را به دست آورید، اما این بار مشخص کنید که می‌خواهید امتیازهای تصمیم‌گیری را به جای پیش‌بینی‌ها دریافت کنید:

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
method="decision_function")
```

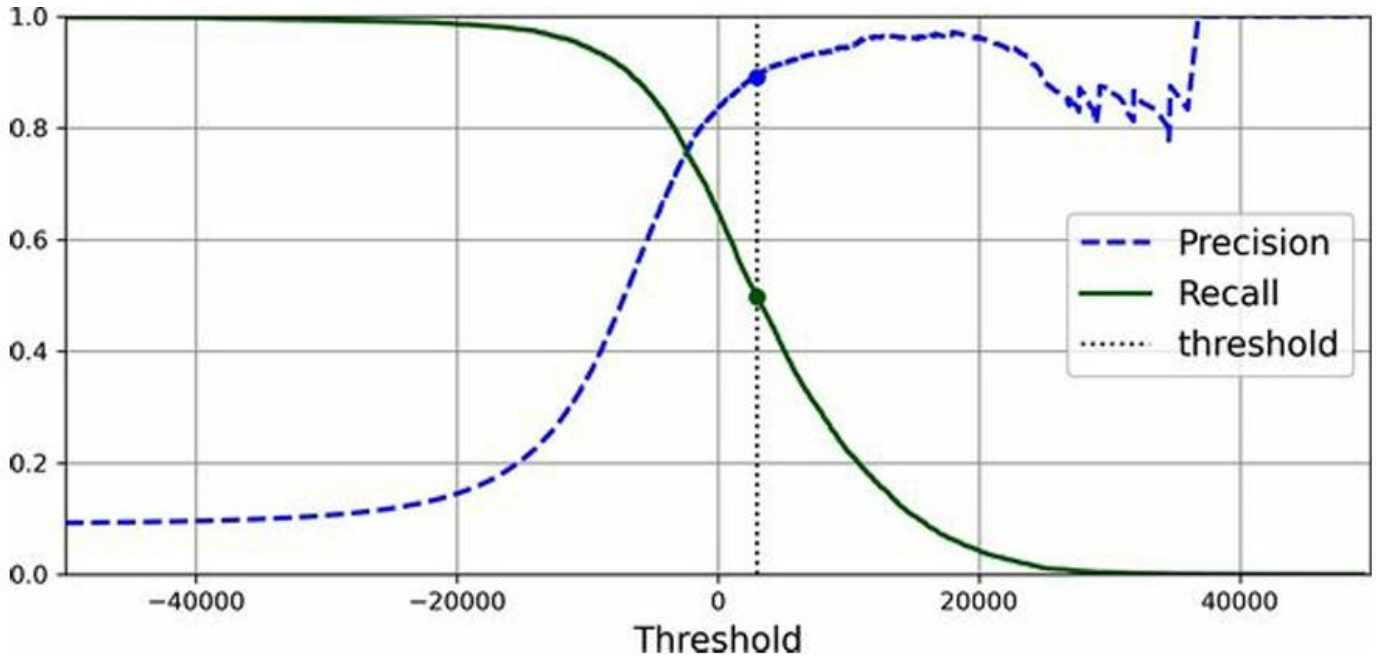
© 2025 Mohsen Zare. All rights reserved.

با این امتیازها، از تابع precision_recall_curve() استفاده کنید تا دقت و recall را برای تمام آستانه‌های ممکن محاسبه کنید (این تابع یک دقت نهایی برابر با ۰ و یک recall نهایی برابر با ۱ اضافه می‌کند که متناظر با آستانه بی‌نهایت است):

```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

در نهایت، از کتابخانه‌ی Matplotlib استفاده کنید تا دقت و recall را به‌صورت تابعی از مقدار آستانه رسم کنید (شکل ۳-۵). همچنین، بیایید آستانه‌ی ۳,۰۰۰ که انتخاب کرده بودیم را در نمودار نشان دهیم:

```
plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
plt.vlines(threshold, 0, 1.0, "k", "dotted", label="threshold")
[...]  # beautify the figure: add grid, legend, axis, labels, and circles
plt.show()
```



شکل ۳-۵. دقت و recall در برابر آستانه تصمیم‌گیری

© 2025 Mohsen Zare. All rights reserved.

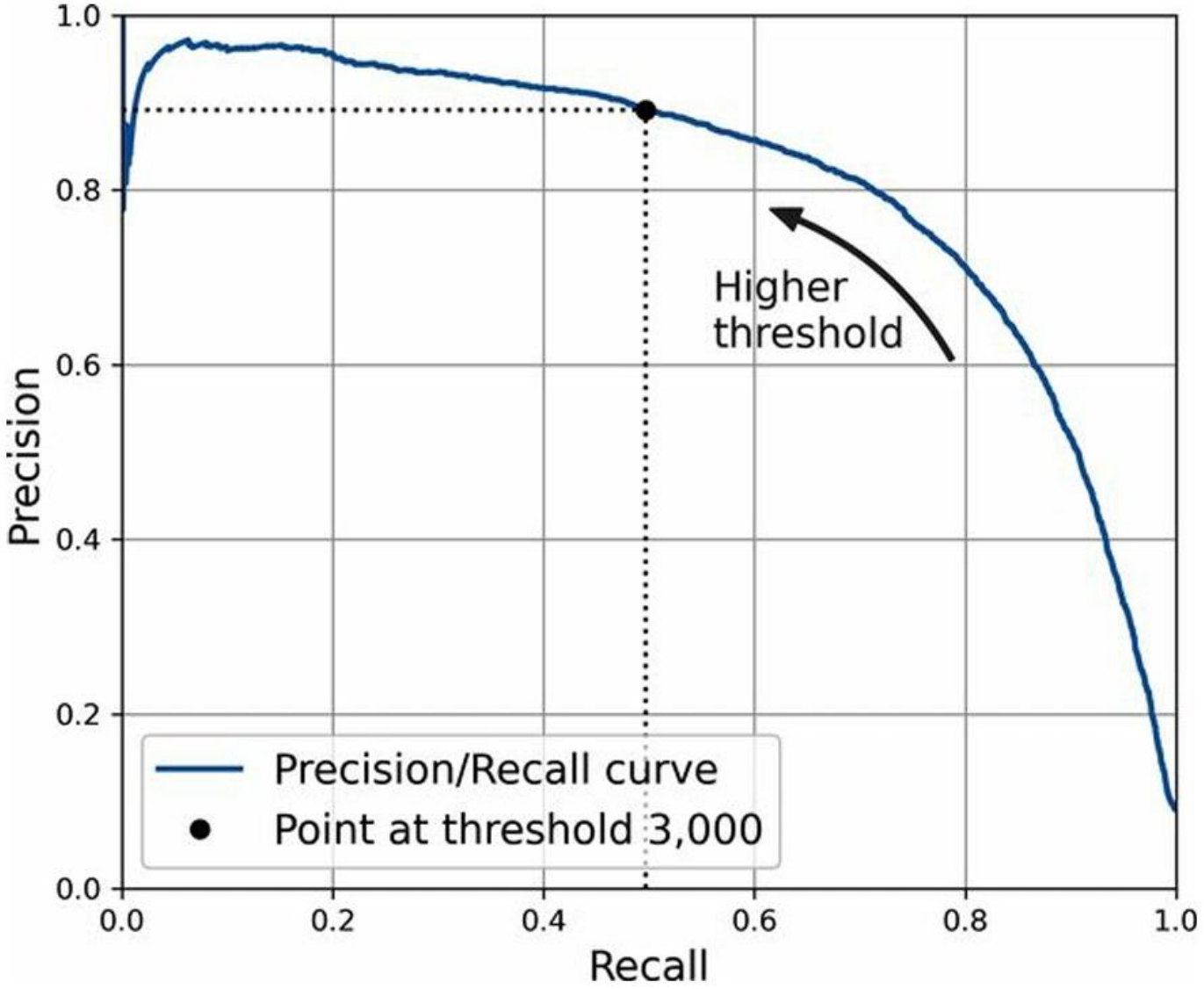
نکته

ممکن است از خود بپرسید چرا منحنی دقت در شکل ۳-۵ نوسان بیشتری نسبت به منحنی recall دارد. دلیل این موضوع آن است که دقت گاهی ممکن است با افزایش آستانه کاهش یابد (اگرچه معمولاً افزایش می‌یابد). برای درک این موضوع، به شکل ۳-۴ بازگردید و ببینید وقتی از آستانه مرکزی فقط یک مقدار به سمت راست حرکت می‌کنید چه اتفاقی می‌افتد: دقت از ۴/۵ (۸۰%) به ۳/۴ (۷۵%) کاهش پیدا می‌کند. در مقابل، recall فقط می‌تواند با افزایش آستانه کاهش یابد، و همین مسئله دلیل صاف بودن منحنیِ آن است.

در این مقدار آستانه، دقت حدود ۹۰% و recall حدود ۵۰% است. یکی دیگر از روش‌های انتخاب یک نقطه تعادل خوب(موازنه) بین دقت و recall، رسم نمودار دقت در برابر recall است، همان‌طور که در شکل ۳-۶ نشان داده شده است (این نقطه آستانه نیز در نمودار مشخص شده است):

```
plt.plot(recalls, precisions, linewidth=2, label="Precision/Recall curve")
[...]  # beautify the figure: add labels, grid, legend, arrow, and text
plt.show()
```

© 2025 Mohsen Zare. All rights reserved.



شکل ۳-۶. دقت در مقابل recall

می‌بینید که دقت از حدود ۸۰% recall به‌سرعت افت می‌کند. بنابراین احتمالاً بخواهید نقطه‌ای را برای تعادل(توازن) بین دقت و recall انتخاب کنید که کمی پیش از این افت شدید قرار دارد—برای مثال، در حدود ۶۰% recall. اما البته این انتخاب کاملاً به پروژه‌ی شما بستگی دارد.

فرض کنید تصمیم بگیرید که به دقت ۹۰% برسید. می‌توانید از نمودار اول برای پیدا کردن آستانه‌ای که باید استفاده شود بهره ببرید، اما این روش خیلی دقیق نیست. در عوض، می‌توانید به‌دنبال کمترین آستانه‌ای باشید که دست‌کم دقت ۹۰% را فراهم می‌کند. برای این کار، می‌توانید از متد ()argmax در آرایه‌ی NumPy استفاده کنید. این متد اولین ایندکس(اندیس) مربوط به مقدار ماکزیمم را بازمی‌گرداند؛ که در این حالت به معنی اولین مقدار True است:

© 2025 Mohsen Zare. All rights reserved.

```
>>> idx_for_90_precision = (precisions >= 0.90).argmax()
>>> threshold_for_90_precision = thresholds[idx_for_90_precision]
>>> threshold_for_90_precision
3370.0194991439557
```

برای انجام پیش‌بینی (فعلاً روی مجموعه آموزش)، به‌جای recall متد ()predict از طبقه‌بند، می‌توانید این کد را اجرا کنید:

```
y_train_pred_90 = (y_scores >= threshold_for_90_precision)
```

بیایید دقت و recall این پیش‌بینی‌ها را بررسی کنیم:

```
>>> precision_score(y_train_5, y_train_pred_90)
0.9000345901072293
>>> recall_at_90_precision = recall_score(y_train_5, y_train_pred_90)
>>> recall_at_90_precision
0.4799852425751706
```

عالیه! شما یک طبقه‌بند با دقت ۹۰% دارید! همان‌طور که می‌بینید، ساختن مدل طبقه‌بندی با تقریباً هر مقدار دقتی که بخواهید کار سختی نیست: فقط کافی‌ست آستانه را به اندازه کافی بالا تنظیم کنید و تمام. اما صبر کنید — نه اینقدر سریع! یک طبقه‌بند با دقت بالا وقتی recall اش خیلی پایین باشد، چندان کاربردی نخواهد بود! برای بسیاری از کاربردها، recall ۴۸% اصلاً قابل قبول نیست.

نکته

اگر کسی گفت: «بیایید به دقت ۹۹% برسیم»، باید بپرسید: «با چه مقدار recall؟»

© 2025 Mohsen Zare. All rights reserved.

منحنی ROC

«منحنیِ مشخصه‌ی عملکرد گیرنده (ROC: *Receiver Operating Characteristic Curve*)» یکی دیگر از ابزارهای رایج برای ارزیابی عملکرد طبقه‌بندهای باینری (Binary Classifiers) است. این منحنی شباهت زیادی به منحنی دقت-recall (Precision/Recall Curve) دارد، با این تفاوت که در آن به‌جای دقت در برابر recall، «نرخ مثبت واقعی (*True Positive Rate* یا همان Recall)» در برابر «نرخ مثبت کاذب (*False Positive Rate* یا FPR)» ترسیم می‌شود. نرخ مثبت کاذب (FPR) که گاهی «*Fall-out*» نیز نامیده می‌شود، نشان‌دهنده نسبت نمونه‌های منفی‌ای است که به اشتباه به‌عنوان مثبت تشخیص داده شده‌اند. این مقدار برابر با ۱ است. در این رابطه، TNR یا «نرخ منفی واقعی (True Negative Rate)» نشان می‌دهد چه نسبتی از نمونه‌های منفی، به‌درستی منفی تشخیص داده شده‌اند. TNR را با نام دیگری نیز می‌شناسند: «ویژگی» (*Specificity*). بنابراین، منحنی ROC در واقع «حساسیت یا *Sensitivity* (یا همان Recall)» را در برابر 1−*specificity* ترسیم می‌کند.

برای رسم منحنی ROC، ابتدا از تابع roc_curve() استفاده می‌کنید تا نرخ مثبت واقعی (TPR) و نرخ مثبت کاذب (FPR) را برای آستانه‌های مختلف محاسبه کنید:

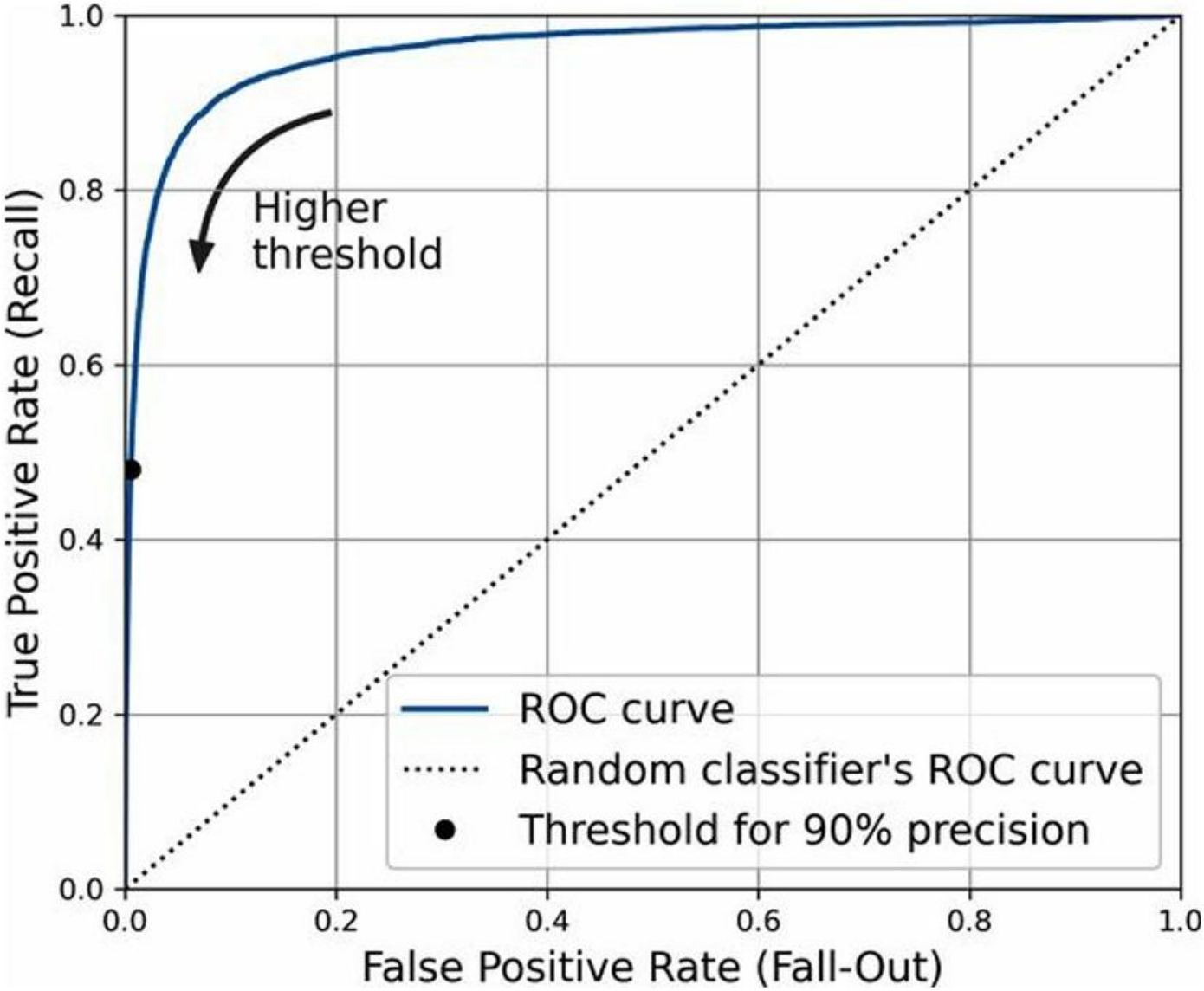
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

سپس می‌توانید مقدار FPR را در برابر TPR با استفاده از Matplotlib رسم کنید. کد زیر نموداری مشابه شکل ۳-۷ تولید می‌کند. برای یافتن نقطه‌ای که متناظر با دقت ۹۰% باشد، باید به دنبال ایندکس آستانه‌ی مورد نظر بگردیم. از آنجایی که فهرست آستانه‌ها در اینجا به ترتیب نزولی مرتب شده، در خط اول از عملگر <= به‌جای >= استفاده می‌کنیم:

```
idx_for_threshold_at_90 = (thresholds <= threshold_for_90_precision).argmax()
tpr_90, fpr_90 = tpr[idx_for_threshold_at_90], fpr[idx_for_threshold_at_90]

plt.plot(fpr, tpr, linewidth=2, label="ROC curve")
plt.plot([0, 1], [0, 1], 'k:', label="Random classifier's ROC curve")
plt.plot([fpr_90], [tpr_90], "ko", label="Threshold for 90% precision")
```

© 2025 Mohsen Zare. All rights reserved.

```
[...]  # beautify the figure: add labels, grid, legend, arrow, and text
plt.show()
```



شکل ۳-۷. منحنی ROC که نرخ مثبت کاذب را در برابر نرخ مثبت واقعی برای تمام آستانه‌های ممکن نشان می‌دهد؛ دایره‌ی مشکی، نقطه‌ی انتخاب‌شده (با دقت ۹۰% و recall ۴۸%) را برجسته می‌کند.

دوباره با یک موازنه روبه‌رو هستیم: هرچه recall (TPR) بالاتر باشد، طبقه‌بند نمونه‌های مثبت کاذب (FPR) بیشتری تولید می‌کند. خط‌چین نشان‌دهندهٔ منحنی ROC یک طبقه‌بند کاملاً تصادفی است؛ یک طبقه‌بند خوب تا حد ممکن باید از این خط فاصله بگیرد (به سمت گوشهٔ بالا-چپ نمودار).

© 2025 Mohsen Zare. All rights reserved.

یکی از روش‌های مقایسه مدل‌های طبقه‌بندی، اندازه‌گیری «مساحت زیر نمودار (AUC: *area under the curve*)» است. یک طبقه‌بند کامل، مقدار ROC AUC برابر با ۱ خواهد داشت، در حالی که یک طبقه‌بند کاملاً تصادفی، مقدار ROC AUC برابر با ۰٫۵ دارد. کتابخانهٔ Scikit-Learn تابعی برای تخمین ROC AUC فراهم کرده است:

```
>>> from sklearn.metrics import roc_auc_score
>>> roc_auc_score(y_train_5, y_scores)
0.96049385540
```

نکته

از آنجا که نمودار ROC شباهت زیادی به منحنیِ دقت-recall (PR) دارد، ممکن است برایتان سوال پیش بیاید که کدام‌یک را باید انتخاب کنید. به‌طور کلی، زمانی که کلاس مثبت نادر است یا زمانی که بیشتر به منفی‌های کاذب اهمیت می‌دهید تا مثبت‌های کاذب، باید از منحنی PR استفاده کنید. در غیر این صورت، از منحنی ROC استفاده کنید. به‌عنوان مثال، با نگاهی به منحنی ROC قبلی (و امتیازِ ROC AUC)، ممکن است فکر کنید که طبقه‌بند شما واقعاً خوب است. اما این بیشتر به این دلیل است که تعداد مثبت‌ها (عدد ۵) نسبت به منفی‌ها (غیر از ۵) کم است. در مقابل، منحنی PR به‌وضوح نشان می‌دهد که طبقه‌بند شما هنوز جای بهبود دارد: نمودار می‌تواند واقعاً به گوشه بالای راست نزدیک‌تر باشد (باز هم به شکل ۶-۳ نگاه کنید).

حال بیایید یک RandomForestClassifier بسازیم که می‌توانیم منحنی PR و امتیاز F_1 آن را با SGDClassifier مقایسه کنیم:

```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(random_state=42)
```

تابع ()precision_recall_curve نیاز به برچسب‌ها و امتیازها برای هر نمونه دارد، بنابراین باید

© 2025 Mohsen Zare. All rights reserved.

RandomForestClassifier را آموزش دهیم و آن را وادار کنیم که برای هر نمونه یک امتیاز اختصاص دهد. اما کلاس RandomForestClassifier به دلیل نحوه عملکردش (که در فصل ۷ توضیح خواهیم داد)، متد decision_function() ندارد. خوشبختانه، این کلاس متد predict_proba() را دارد که احتمال کلاس‌ها را برای هر نمونه باز می‌گرداند و می‌توانیم از احتمال کلاس مثبت به‌عنوان امتیاز استفاده کنیم، بنابراین این روش به خوبی کار خواهد کرد. ما می‌توانیم از تابع cross_val_predict() برای آموزش RandomForestClassifier با استفاده از اعتبارسنجی متقاطع و پیش‌بینی احتمال کلاس‌ها برای هر تصویر به‌صورت زیر استفاده کنیم:

```
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method="predict_proba")
```

بیایید نگاهی به احتمال کلاس‌ها برای دو تصویر اول از مجموعه آموزش بیندازیم:

```
>>> y_probas_forest[:2]
array([[0.11, 0.89],
       [0.99, 0.01]])
```

مدل پیش‌بینی می‌کند که تصویر اول با احتمال ۸۹% مثبت است و تصویر دوم را با احتمال ۹۹% منفی پیش‌بینی می‌کند. از آن‌جا که هر تصویر یا مثبت است یا منفی، احتمال‌ها در هر ردیف مجموعاً برابر با ۱۰۰% می‌شوند.

هشدار

این‌ها احتمال‌های تخمین زده شده هستند، نه احتمال‌های واقعی. به عنوان مثال، اگر به تمام تصاویری که مدل آن‌ها را به عنوان مثبت با احتمال تخمینی بین ۵۰٪ تا ۶۰٪ طبقه‌بندی کرده است نگاه کنید، تقریباً ۹۴٪ از آن‌ها واقعاً مثبت هستند. بنابراین، احتمال‌های تخمینی مدل در این مورد خیلی پایین بوده‌اند—اما مدل‌ها می‌توانند بیش از حد مطمئن هم باشند. بسته sklearn.calibration ابزارهایی برای کالیبره کردن احتمال‌های تخمینی و نزدیک کردن آن‌ها به احتمال‌های واقعی دارد. برای جزئیات بیشتر به بخش مطالب اضافی در نوت‌بوک این فصل مراجعه کنید.

© 2025 Mohsen Zare. All rights reserved.

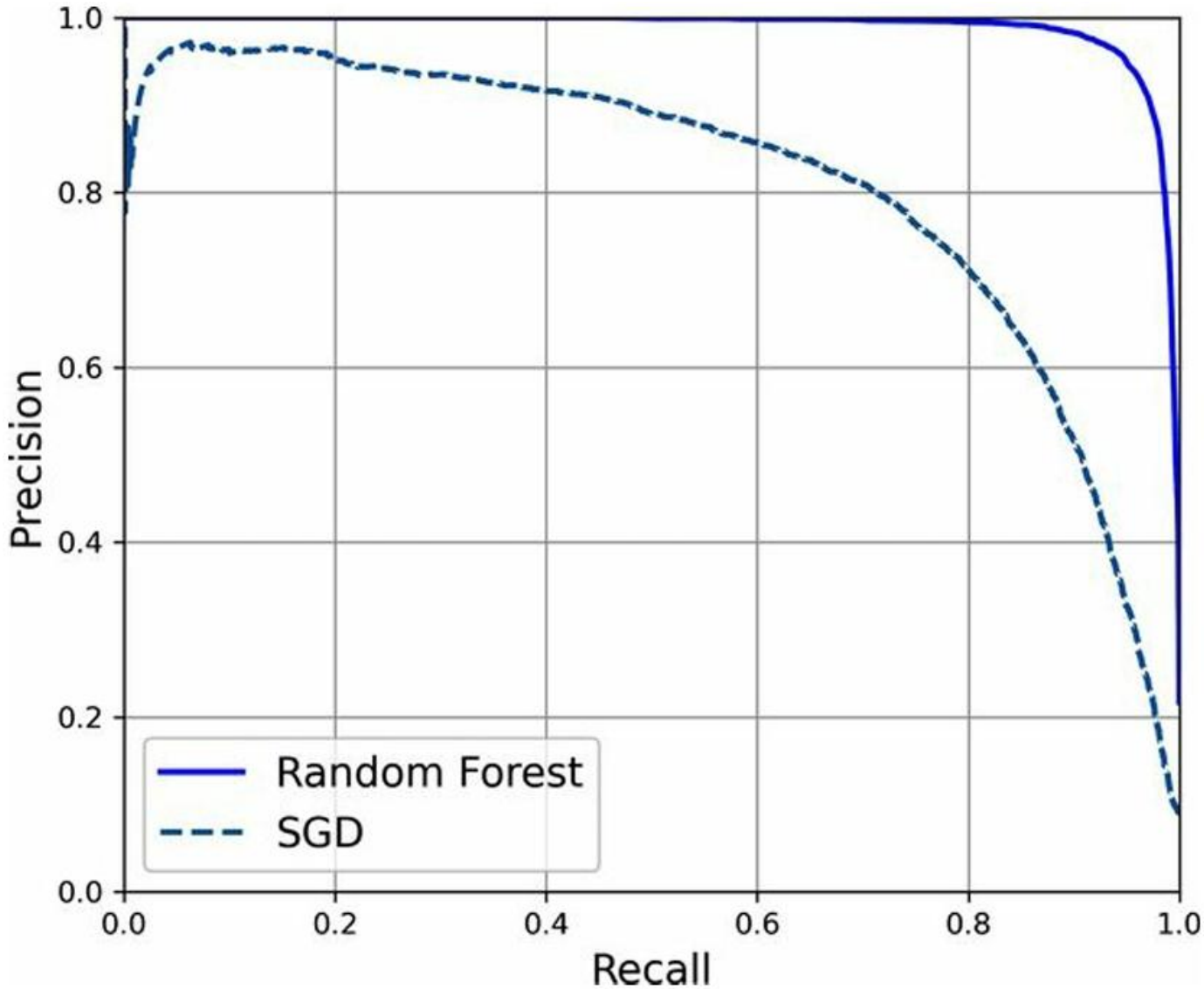
ستون دوم احتمال‌های تخمینی برای کلاس مثبت را شامل می‌شود، بنابراین این‌ها را به تابع precision_recall_curve() می‌دهیم:

```
y_scores_forest = y_probas_forest[:, 1]
precisions_forest, recalls_forest, thresholds_forest = precision_recall_curve(
    y_train_5, y_scores_forest)
```

حال آماده‌ایم تا منحنی PR را رسم کنیم. مقایسه منحنی PR اول با این منحنی مفید است تا تفاوت‌ها را ببینیم (شکل ۳-۸):

```
plt.plot(recalls_forest, precisions_forest, "b-", linewidth=2,
       label="Random Forest")
plt.plot(recalls, precisions, "--", linewidth=2, label="SGD")
[...]  # beautify the figure: add labels, grid, and legend
plt.show()
```

© 2025 Mohsen Zare. All rights reserved.



شکل ۳-۸. مقایسه منحنی‌های PR: طبقه‌بند جنگل تصادفی از طبقه‌بند SGD برتر است زیرا منحنی PR آن بسیار نزدیک‌تر به گوشهٔ بالا-راست است و AUC بیشتری دارد

همانطور که در شکل ۳-۸ مشاهده می‌کنید، منحنی PR طبقه‌بند جنگل تصادفی (RandomForestClassifier) به مراتب بهتر از منحنی PR طبقه‌بند SGD است: این منحنی بسیار نزدیک‌تر به گوشه بالا-راست قرار دارد. امتیاز F_1 و امتیاز ROC AUC آن نیز به‌طور قابل توجهی بهتر است:

```
>>> y_train_pred_forest = y_probas_forest[:, 1] >= 0.5  # positive proba ≥ 50%
>>> f1_score(y_train_5, y_pred_forest)
0.9242275142688446
```

© 2025 Mohsen Zare. All rights reserved.

```
>>> roc_auc_score(y_train_5, y_scores_forest)
0.9983436731328145
```

تلاش کنید که دقت و recall را اندازه‌گیری کنید: باید حدود ۹۹٫۱ % دقت و ۸۶٫۶ % recall به‌دست آورید. بد نیست!

حالا شما می‌دانید که چگونه طبقه‌بندهای باینری را آموزش دهید، معیار مناسب برای کارتان را انتخاب کنید، طبقه‌بندهای خود را با استفاده از اعتبارسنجی متقاطع ارزیابی کنید، موازنه‌ی دقت-recall که با نیازهای شما همخوانی دارد را انتخاب کنید و از چندین معیار و منحنی برای مقایسه مدل‌های مختلف استفاده کنید. حالا آماده‌اید تا سعی کنید تشخیص دهید که بیشتر از فقط «عدد ۵» چه چیزی وجود دارد.

© 2025 Mohsen Zare. All rights reserved.

طبقه‌بندی چندکلاسه

در حالی که «طبقه‌بندهای باینری» دو کلاس را از یکدیگر تشخیص می‌دهند، «طبقه‌بندهای چندکلاسه که با نام های *multiclass classifiers* یا *multinomial classifiers*» شناخته می‌شوند قادرند بیش از دو کلاس را از یکدیگر تفکیک کنند.

برخی از طبقه‌بندهای Scikit-Learn (مثلِ LogisticRegression، RandomForestClassifier و GaussianNB) قادر به پردازش چندکلاسه به‌طور مستقیم هستند. اما برخی دیگر به‌طور خاص طبقه‌بندهای باینری هستند (مثل SGDClassifier و SVC). با این حال، استراتژی‌های مختلفی وجود دارند که می‌توانند برای انجام طبقه‌بندی چندکلاسه با استفاده از چندین طبقه‌بند باینری به‌کار گرفته شوند.

یکی از روش‌ها برای ایجاد سیستمی که بتواند تصاویر ارقام را به ۱۰ کلاس (از ۰ تا ۹) دسته‌بندی کند، این است که ۱۰ طبقه‌بند باینری آموزش داده شود، یکی برای هر رقم (مثلاً یک آشکارساز برای ۰، یکی برای ۱، یکی برای ۲ و به همین ترتیب). سپس، زمانی که می‌خواهید یک تصویر را طبقه‌بندی کنید، امتیاز تصمیم‌گیری را از هر یک از این طبقه‌بندها برای آن تصویر دریافت می‌کنید و کلاسی را انتخاب می‌کنید که طبقه‌بندِ مربوط به آن، بالاترین امتیاز را داده باشد. به این روش، استراتژی «یکی در برابر بقیه (*One-versus-the-Rest* یا OvR)» گفته می‌شود که گاهی «یکی در برابر همه (*One-versus-All* یا OvA)» نیز نامیده می‌شود.

استراتژی دیگر این است که برای هر جفت از ارقام، یک طبقه‌بند باینری آموزش دهید: یکی برای تشخیص بین ۰ و ۱، دیگری برای ۰ و ۲، یکی دیگر برای ۱ و ۲، و به همین ترتیب. این روش به «یکی در برابر یکی (*One-versus-One* یا OvO)» معروف است. اگر N کلاس داشته باشید، باید $N \times (N-1) / 2$ طبقه‌بند آموزش دهید. برای مسئله MNIST (که شامل ۱۰ رقم است)، این یعنی آموزش ۴۵ طبقه‌بند باینری! زمانی که می‌خواهید یک تصویر را طبقه‌بندی کنید، باید آن را از میان هر ۴۵ طبقه‌بند عبور دهید و بررسی کنید که کدام کلاس در بیشترین «دوئل» برنده شده است. مزیت اصلی روش OvO این است که هر طبقه‌بند فقط روی بخشی از داده‌های آموزشی که مربوط به دو کلاس موردنظرش است آموزش می‌بیند، و نه روی کل مجموعه داده.

© 2025 Mohsen Zare. All rights reserved.

برخی الگوریتم‌ها (مانند طبقه‌بندهای ماشین بردار پشتیبان یا SVM) با افزایش اندازه مجموعه‌ی آموزش، مقیاس‌پذیری ضعیفی دارند. برای این الگوریتم‌ها، استفاده از روش OvO (یکی در برابر یکی) ترجیح داده می‌شود، چون آموزش تعداد زیادی طبقه‌بند روی مجموعه داده‌های کوچک سریع‌تر است از آموزش تعداد کمی طبقه‌بند روی مجموعه داده‌های بزرگ. با این حال، برای بیشتر الگوریتم‌های طبقه‌بندی باینری، روش OvR (یکی در برابر بقیه) معمولاً انتخاب بهتری است.

کتابخانه Scikit-Learn زمانی که شما تلاش کنید از یک الگوریتم طبقه‌بندی باینری برای یک مسئله‌ی چندکلاسی استفاده کنید، به‌صورت خودکار یکی از دو استراتژی OvR یا OvO را بسته به نوع الگوریتم اعمال می‌کند. بیایید این موضوع را با یک طبقه‌بند ماشین بردار پشتیبان (SVM) آزمایش کنیم و از کلاس sklearn.svm.SVC استفاده کنیم (جزئیات این کلاس در فصل ۵ توضیح داده شده است). برای اینکه زمان آموزش بیش از حد طولانی نشود، فقط روی ۲,۰۰۰ تصویر اول آموزش می‌دهیم:

```
from sklearn.svm import SVC

svm_clf = SVC(random_state=42)
svm_clf.fit(X_train[:2000], y_train[:2000])  # y_train, not y_train_5
```

این کار واقعاً ساده بود! ما مدل SVC را با استفاده از برچسب‌های اصلی هدف، یعنی اعداد ۰ تا ۹ (y_train) آموزش دادیم، نه با برچسب‌های مسئله‌ی «۵ در برابر بقیه» (y_train_5). از آنجا که در این مسئله ۱۰ کلاس وجود دارد (یعنی بیش از دو کلاس)، Scikit-Learn به‌طور خودکار از استراتژی OvO استفاده کرد و ۴۵ طبقه‌بند باینری آموزش داد. حالا بیایید روی یک تصویر، پیش‌بینی انجام دهیم:

```
>>> svm_clf.predict([some_digit])
array(['5'], dtype=object)
```

دقیقاً همین‌طور است! این کد در واقع ۴۵ پیش‌بینی انجام داده—یعنی یک پیش‌بینی برای هر جفت کلاس—و در نهایت کلاسی را انتخاب کرده که بیشترین تعداد پیروزی را در این دوئل‌ها به‌دست آورده است. اگر متد decision_function() را فراخوانی کنید، می‌بینید که این متد برای هر نمونه ۱۰ امتیاز برمی‌گرداند—یکی برای هر کلاس. هر کلاس امتیازی برابر با تعداد دوئل‌های برنده‌شده می‌گیرد، به‌علاوه یا منهای یک اصلاح جزئی (حداکثر ±۰٫۳۳) برای شکستن تساوی‌ها، که این مقدار بر اساس امتیازات خروجی طبقه‌بند محاسبه می‌شود:

```
>>> some_digit_scores = svm_clf.decision_function([some_digit])
```

© 2025 Mohsen Zare. All rights reserved.

```
>>> some_digit_scores.round(2)
array([[ 3.79,  0.73,  6.06,  8.3 , -0.29,  9.3 ,  1.75,  2.77,  7.21,
        4.82]])
```

بیشترین امتیاز ۹٫۳ است، و همان‌طور که انتظار می‌رود، این امتیاز مربوط به کلاس ۵ است:

```
>>> class_id = some_digit_scores.argmax()
>>> class_id
5
```

وقتی یک طبقه‌بند آموزش می‌بیند، لیست برچسب‌های کلاس هدف را در ویژگی classes_ ذخیره می‌کند که به ترتیب مقدار مرتب شده‌اند. در مورد مجموعه داده MNIST، اندیس هر کلاس در آرایه classes_ به‌طور خوش‌شانس‌گونه‌ای دقیقاً با شماره کلاس یکی است (مثلاً کلاس در اندیس ۵ همان کلاس '۵' است). اما به طور کلی این اتفاق همیشه نمی‌افتد و باید برچسب کلاس را به این شکل پیدا کنید:

```
>>> svm_clf.classes_
array(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'], dtype=object)
>>> svm_clf.classes_[class_id]
'5'
```

اگر بخواهید در Scikit-Learn به‌صورت اجباری از روش «یکی در برابر یکی» یا «یکی در برابر بقیه» استفاده کنید، می‌توانید از کلاس‌های OneVsOneClassifier یا OneVsRestClassifier بهره ببرید. کافی است یک نمونه از این کلاس‌ها ایجاد کرده و یک دسته‌بند(Classifier) را به سازنده‌ی آن بدهید—این طبقه‌بند حتی لازم نیست الزاماً باینری باشد. برای نمونه، می‌توان با استفاده از یک SVC، یک طبقه‌بند چندکلاسه مبتنی بر استراتژی OvR ایجاد کرد:

```
from sklearn.multiclass import OneVsRestClassifier

ovr_clf = OneVsRestClassifier(SVC(random_state=42))
ovr_clf.fit(X_train[:2000], y_train[:2000])
```

بیایید یک پیش‌بینی انجام دهیم و همچنین بررسی کنیم که چند طبقه‌بند در این فرایند آموزش داده شده‌اند:

```
>>> ovr_clf.predict([some_digit])
```

© 2025 Mohsen Zare. All rights reserved.

```
array(['5'], dtype='<U1')
>>> len(ovr_clf.estimators_)
10
```

آموزش یک SGDClassifier بر روی یک مجموعه داده چندکلاسی و استفاده از آن برای انجام پیش‌بینی نیز به همان سادگی است:

```
>>> sgd_clf = SGDClassifier(random_state=42)
>>> sgd_clf.fit(X_train, y_train)
>>> sgd_clf.predict([some_digit])
array(['3'], dtype='<U1')
```

اوه، اشتباه شد. خطا در پیش‌بینی رخ می‌دهد! این بار Scikit-Learn در پس‌زمینه از استراتژی OvR استفاده کرد: از آنجا که ۱۰ کلاس وجود داشت، ده طبقه‌بند باینری آموزش داد. متد decision_function() اکنون برای هر کلاس یک مقدار بازمی‌گرداند. بیایید امتیازهایی را که طبقه‌بند SGD به هر کلاس اختصاص داده است بررسی کنیم:

```
>>> sgd_clf.decision_function([some_digit]).round()
array([[-31893., -34420.,  -9531.,   1824., -22320.,  -1386., -26189.,
        -16148.,  -4604., -12051.]])
```

می‌بینید که طبقه‌بند چندان به پیش‌بینی خود مطمئن نیست: تقریباً همه‌ی امتیازها بسیار منفی‌اند، در حالی که کلاس ۳ امتیاز ۱٫۸۲۴+ دارد و کلاس ۵ هم با امتیاز ۱٫۳۸۶- فاصلهٔ زیادی ندارد. البته بهتر است این طبقه‌بند را فقط روی یک تصویر ارزیابی نکنید. از آنجا که تعداد تصاویر در هر کلاس تقریباً برابر است، معیار دقت مناسب خواهد بود. همان‌طور که همیشه، می‌توانید از تابع cross_val_score() برای ارزیابی مدل استفاده کنید:

```
>>> cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
array([0.87365, 0.85835, 0.8689 ])
```

این مدل در تمام دسته‌های اعتبارسنجی متقاطع دقتی بالاتر از ۸۵٫۸ % به‌دست آورده است. اگر از یک طبقه‌بند کاملاً تصادفی استفاده می‌کردید، تنها حدود ۱۰ % دقت داشتید، بنابراین این نتیجه بد نیست؛ اما هنوز جای پیشرفت زیادی وجود دارد. فقط با مقیاس‌بندی ورودی‌ها (همان‌طور که در فصل ۲ توضیح داده شد) می‌توانید دقت را به بیش از ۸۹٫۱ % برسانید:

© 2025 Mohsen Zare. All rights reserved.

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> X_train_scaled = scaler.fit_transform(X_train.astype("float64"))
>>> cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
array([0.8983, 0.891 , 0.9018])
```

© 2025 Mohsen Zare. All rights reserved.

تحلیل خطا

اگر این یک پروژه واقعی بود، در این مرحله از کار، باید مراحل موجود در چک‌لیست پروژه یادگیری ماشین (ضمیمهٔ(پیوست) الف (A)) را دنبال می‌کردید: بررسی روش‌های آماده‌سازی داده، آزمایش مدل‌های مختلف، انتخاب بهترین‌ها، تنظیم دقیق هایپرپارامترها با GridSearchCV و خودکارسازی هر چه بیشتر فرایند. اما در اینجا فرض می‌کنیم که شما یک مدل امیدوارکننده پیدا کرده‌اید و حالا به دنبال روش‌هایی برای بهبود آن هستید. یکی از راه‌ها، تحلیل نوع خطاهایی است که مدل مرتکب می‌شود.

ابتدا باید به ماتریس درهم‌ریختگی نگاه کنید. برای این کار، ابتدا با استفاده از تابع cross_val_predict() پیش‌بینی‌ها را انجام می‌دهید؛ سپس برچسب‌های واقعی و برچسب‌های پیش‌بینی‌شده را به تابع confusion_matrix() می‌دهید، درست مانند کاری که قبلاً برای حالت باینری انجام دادید. البته حالا که به‌جای ۲ کلاس، ۱۰ کلاس داریم، ماتریس درهم‌ریختگی شامل تعداد زیادی عدد خواهد بود و خواندن آن ممکن است دشوار شود.

یک نمودار رنگی از ماتریس درهم‌ریختگی، تحلیل آن را بسیار ساده‌تر می‌کند. برای رسم چنین نموداری می‌توانید از تابع ConfusionMatrixDisplay.from_predictions() به صورت زیر استفاده کنید:

```
from sklearn.metrics import ConfusionMatrixDisplay

y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred)
plt.show()
```

این کد نمودار سمت چپ در شکل ۳-۹ را ایجاد می‌کند. این ماتریس درهم‌ریختگی در نگاه اول خوب به نظر می‌رسد: بیشتر تصاویر روی قطر اصلی قرار دارند، به این معنی که به‌درستی طبقه‌بندی شده‌اند. توجه کنید که خانه(سلول) روی قطر در سطر شماره ۵ و ستون شماره ۵ کمی تیره‌تر از سایر ارقام است. این موضوع می‌تواند به این دلیل باشد که مدل در تشخیص رقم ۵ خطاهای بیشتری داشته، یا اینکه تعداد رقم ۵ در مجموعه داده کمتر از سایر ارقام است. به همین دلیل مهم است که ماتریس درهم‌ریختگی را نرمال‌سازی کنیم؛ یعنی هر مقدار را بر تعداد کل تصاویر در کلاس واقعی(true) مربوطه

© 2025 Mohsen Zare. All rights reserved.

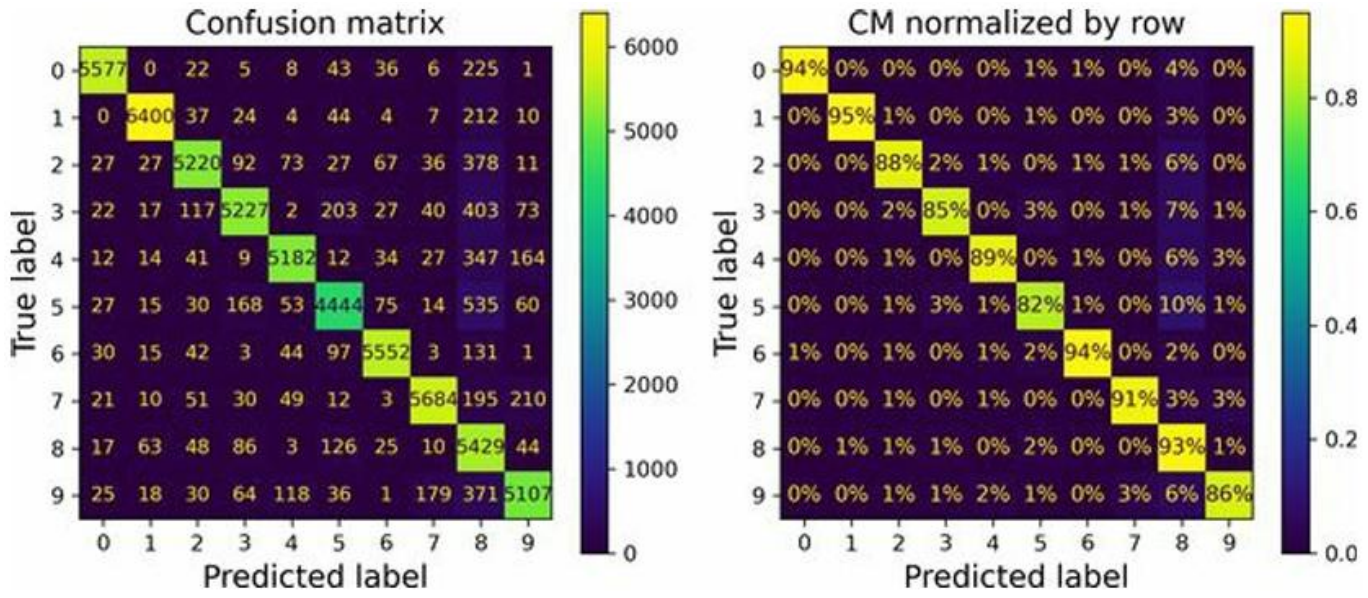
تقسیم کنیم (یعنی تقسیم بر مجموع سطر). این کار به‌سادگی با تنظیم پارامتر "normalize="true انجام می‌شود. همچنین می‌توانیم با آرگومان "values_format=".0% مقادیر را به صورت درصد بدون اعشار نمایش دهیم. کد زیر نمودار سمت راست شکل ۳-۹ را تولید می‌کند:

```
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred,
                                        normalize="true", values_format=".0%")
plt.show()
```

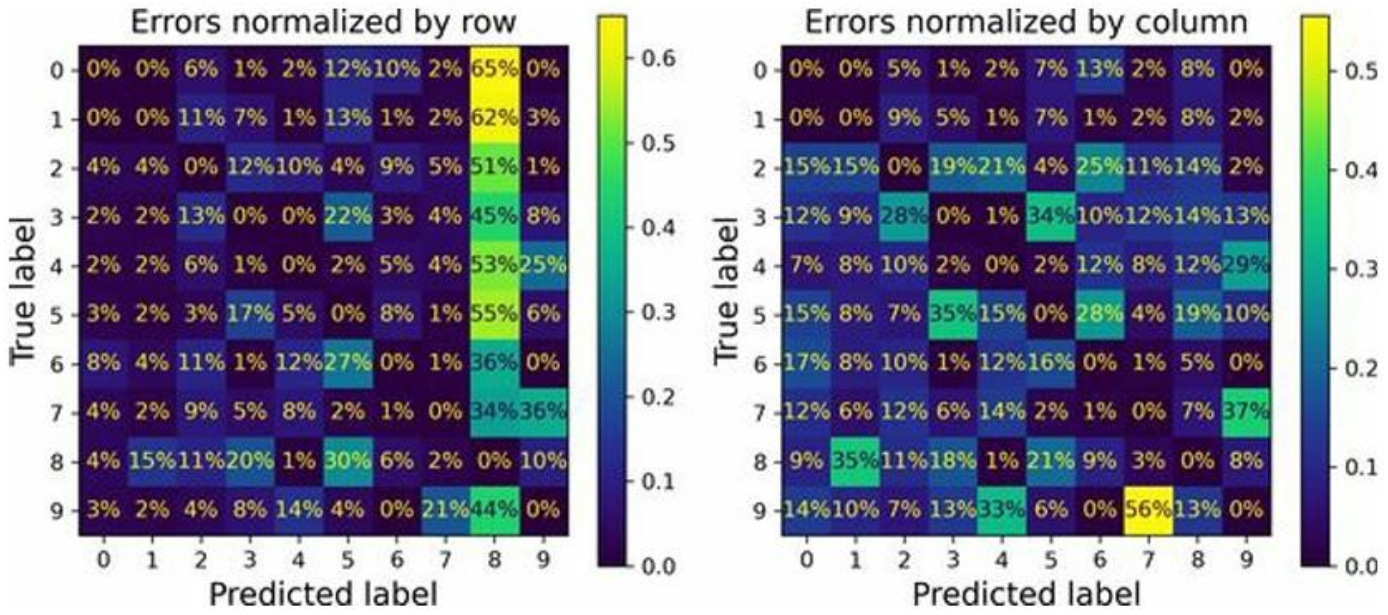
حالا می‌توانیم به‌وضوح ببینیم که تنها ۸۲% از تصاویر رقم ۵ به‌درستی طبقه‌بندی شده‌اند. رایج‌ترین خطایی که مدل در مورد تصاویر رقم ۵ مرتکب شده، طبقه‌بندی اشتباه آن‌ها به‌عنوان رقم ۸ بوده است: این اتفاق برای ۱۰% از تمام نمونه‌های رقم ۵ افتاده است. اما تنها ۲% از تصاویر رقم ۸ به‌اشتباه به‌عنوان رقم ۵ شناسایی شده‌اند؛ ماتریس‌های درهم‌ریختگی معمولاً متقارن نیستند! اگر با دقت نگاه کنید، متوجه می‌شوید که بسیاری از ارقام به‌عنوان رقم ۸ اشتباه طبقه‌بندی شده‌اند، اما این موضوع در این نمودار به‌وضوح دیده نمی‌شود. اگر بخواهید خطاها بیشتر به چشم بیایند، می‌توانید وزن پیش‌بینی‌های درست را صفر کنید. کد زیر دقیقاً همین کار را انجام می‌دهد و نمودار سمت چپ شکل ۳-۱۰ را تولید می‌کند:

```
sample_weight = (y_train_pred != y_train)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred,
                                        sample_weight=sample_weight,
                                        normalize="true", values_format=".0%")
plt.show()
```

© 2025 Mohsen Zare. All rights reserved.



شکل ۳-۹. ماتریس درهم‌ریختگی (سمت چپ) و همان ماتریس نرمال‌شدهٔ CM بر اساس سطرها (راست)



شکل ۳-۱۰. ماتریس درهم‌ریختگی شامل فقط خطاها، نرمال‌شده بر اساس سطرها (سمت چپ) و بر اساس ستون‌ها (سمت راست)

حالا می‌توانید خیلی واضح‌تر انواع خطاهایی را که طبقه‌بند مرتکب می‌شود، ببینید. ستون مربوط به کلاس ۸ اکنون واقعاً روشن است که تأیید می‌کند بسیاری از تصاویر به‌اشتباه به‌عنوان رقم ۸ شناسایی شده‌اند. در واقع، این رایج‌ترین اشتباه طبقه‌بندی برای تقریباً همه کلاس‌هاست. اما در تفسیر درصدهای این نمودار باید احتیاط کنید: به یاد داشته باشید که پیش‌بینی‌های درست را حذف کرده‌ایم.

© 2025 Mohsen Zare. All rights reserved.

برای مثال، عدد ۳۶% در سطر ۷ و ستون ۹ به این معنا «نیست» که ۳۶% از تمام تصاویر رقم ۷ به‌اشتباه به‌عنوان رقم ۹ شناسایی شده‌اند، بلکه یعنی ۳۶% از «خطاهایی (*errors*)» که مدل در مورد تصاویر رقم ۷ مرتکب شده، اشتباه در تشخیص آن‌ها به‌عنوان رقم ۹ بوده است. در واقعیت، فقط ۳% از تصاویر رقم ۷ به‌عنوان رقم ۹ اشتباه شناسایی شده‌اند، همان‌طور که در نمودار سمت راست شکل ۳-۹ می‌توانید ببینید.

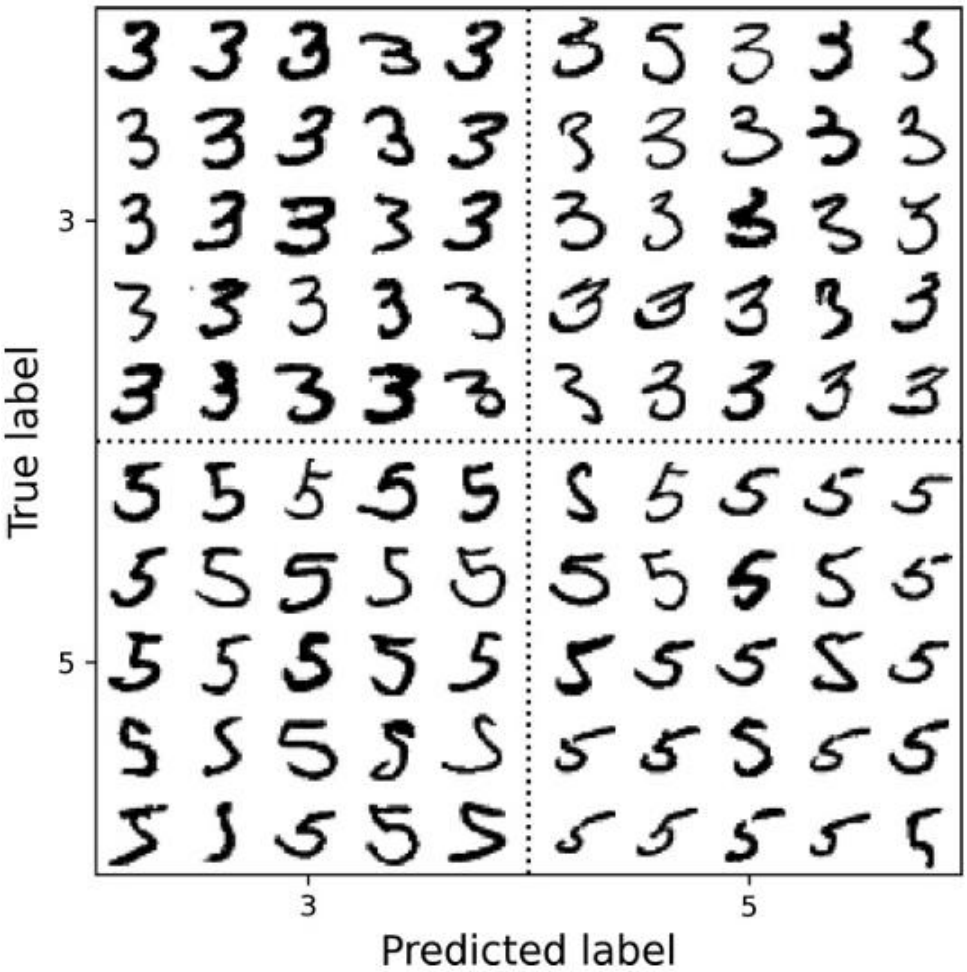
همچنین می‌توان ماتریس درهم‌ریختگی را به‌جای نرمال‌سازی بر اساس سطر، بر اساس ستون نرمال کرد: اگر "normalize="pred تنظیم شود، نمودار سمت راست شکل ۳-۱۰ به‌دست می‌آید. برای مثال، می‌توانید ببینید که ۵۶% از اعداد ۷ که اشتباه طبقه‌بندی شده‌اند، در واقع ۹ بوده‌اند.

تحلیل ماتریس درهم‌ریختگی معمولاً دیدگاه‌هایی (insights) برای بهبود طبقه‌بند ارائه می‌دهد. با نگاه به این نمودارها، به نظر می‌رسد باید تلاش خود را روی کاهش اشتباهات مربوط به عدد ۸ بگذارید. برای مثال، می‌توانید داده‌های آموزشی بیشتری برای اعدادی که شبیه ۸ هستند (اما واقعاً ۸ نیستند) جمع‌آوری کنید تا طبقه‌بند بتواند آن‌ها را از ۸ واقعی تشخیص دهد. یا می‌توانید ویژگی‌های جدیدی طراحی کنید که به طبقه‌بند کمک کند؛ مثلاً نوشتن الگوریتمی برای شمارش تعداد حلقه‌های بسته (مثلاً عدد ۸ دو حلقه، ۶ یک حلقه و ۵ هیچ حلقه‌ای ندارد). همچنین می‌توانید تصاویر را پیش‌پردازش کنید (با استفاده از کتابخانه‌هایی مثل Scikit-Image، Pillow یا OpenCV) تا برخی الگوها مثل حلقه‌های بسته بهتر دیده شوند.

تحلیل خطاهای فردی نیز می‌تواند روش خوبی برای فهمیدن عملکرد طبقه‌بند و دلایل اشتباهات آن باشد. برای مثال، بیایید نمونه‌هایی از اعداد ۳ و ۵ را به سبک ماتریس درهم‌ریختگی رسم کنیم (شکل ۳-۱۱):

```
cl_a, cl_b = '3', '5'
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]
[...]  # plot all images in X_aa, X_ab, X_ba, X_bb in a confusion matrix style
```

© 2025 Mohsen Zare. All rights reserved.



شکل ۳-۱۱. چند تصویر از اعداد ۳ و ۵ که به صورت یک ماتریس درهم‌ریختگی مرتب شده‌اند

همان‌طور که می‌بینید، بعضی از اعدادی که طبقه‌بندی‌کننده اشتباه تشخیص داده (یعنی در بلوک‌های پایین چپ و بالا راست) آن‌قدر بد نوشته شده‌اند که حتی برای انسان هم تشخیص آن‌ها سخت است. با این حال، بیشتر تصاویر اشتباه طبقه‌بندی‌شده برای ما خطاهای واضحی به نظر می‌رسند. ممکن است فهمیدن دلیل اشتباهاتِ طبقه‌بند دشوار باشد، اما به خاطر داشته باشید که مغز انسان یک سیستم فوق‌العاده برای تشخیص الگو است و سیستم بینایی ما پیش‌پردازش‌های پیچیده‌ای انجام می‌دهد قبل از اینکه اطلاعات به سطح آگاهی ما برسد. بنابراین، این‌که این کار ساده به نظر می‌رسد به معنای آسان بودن آن نیست. به یاد داشته باشید که ما از یک SGDClassifier ساده استفاده کردیم که در واقع یک مدل خطی است: کاری که انجام می‌دهد این است که برای هر کلاس به هر پیکسل یک وزن اختصاص می‌دهد و وقتی تصویر جدیدی می‌بیند، مجموع شدت پیکسل‌های وزن‌دار را محاسبه می‌کند تا امتیازی برای هر کلاس به دست آورد. از آن‌جا که اعداد ۳ و ۵ فقط در چند پیکسل با هم تفاوت دارند، این مدل به‌راحتی آن‌ها را با هم اشتباه می‌گیرد.

© 2025 Mohsen Zare. All rights reserved.

تفاوت اصلی بین اعداد ۳ و ۵(3, 5) در موقعیت خط کوچکی است که خط بالایی را به قوس پایینی وصل می‌کند. اگر عدد ۳(3) را طوری بکشید که این اتصال کمی به سمت چپ جابجا شده باشد، مدل ممکن است آن را به عنوان ۵(5) تشخیص دهد و بالعکس. به عبارت دیگر، این مدل نسبت به جابجایی و چرخش تصویر حساس است. یکی از راه‌های کاهش اشتباه بین ۳ و ۵، پیش‌پردازش تصاویر برای اطمینان از قرارگیری صحیح مرکز آن‌ها و عدم چرخش زیاد است. با این حال، این کار سخت است چون باید چرخش درست هر تصویر پیش‌بینی شود. رویکرد ساده‌تر این است که مجموعه داده‌های آموزش را با تصاویر کمی جابجا شده و چرخیده شده افزایش دهیم. این باعث می‌شود مدل نسبت به این تغییرات تحمل بیشتری پیدا کند. این روش را «افزایش داده ($data\ augmentation$)» می‌نامند— تولید داده‌های مصنوعی یا تکمیلی؛ تکنیکی برای افزایش تعداد و تنوع داده‌ها بدون جمع‌آوری داده‌های جدید، با تغییرات کوچک مثل چرخش، برش، نویز، تغییر رنگ یا ترجمه متن— که در فصل ۱۴ به آن پرداخته خواهد شد؛ همچنین تمرین ۲ در پایان همین فصل را ببینید.

© 2025 Mohsen Zare. All rights reserved.

طبقه‌بندی چندبرچسبی

تا اینجا، هر نمونه فقط به یک کلاس اختصاص داده شده بود. اما در بعضی موارد ممکن است بخواهید مدل شما چند کلاس مختلف را برای هر نمونه پیش‌بینی کند. فرض کنید یک مدلِ طبقه‌بندِ تشخیص چهره دارید: اگر در یک تصویر چند نفر را شناسایی کند، باید برای هر نفر یک برچسب جداگانه بگذارد. مثلاً فرض کنید مدل برای سه چهره آموزش دیده است: آلیس، باب و چارلی. حال اگر تصویری شامل آلیس و چارلی باشد، خروجی باید به صورت [True, False, True] باشد (یعنی «آلیس بله، باب نه، چارلی بله»). چنین سیستمی که چند برچسب باینری را همزمان برای هر نمونه پیش‌بینی می‌کند، «سیستم طبقه‌بندی چندبرچسبی (*multilabel classification*)» نام دارد.

فعلاً وارد موضوع تشخیص چهره نمی‌شویم، اما بیایید یک مثال ساده‌تر را فقط برای توضیح بررسی کنیم:

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= '7')
y_train_odd = (y_train.astype('int8') % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

این کد آرایه‌ای به نام y_multilabel ایجاد می‌کند که برای هر تصویر عدد، دو برچسب هدف دارد: اولین برچسب نشان می‌دهد که آیا عدد بزرگ است (۷، ۸ یا ۹) یا نه، و برچسب دوم مشخص می‌کند که آیا عدد فرد است یا خیر. سپس کد یک نمونه از KNeighborsClassifier می‌سازد که از طبقه‌بندی چندبرچسبی پشتیبانی می‌کند (همه طبقه‌بندها این قابلیت را ندارند) و این مدل را با استفاده از آرایه اهداف چندگانه(multiple targets) آموزش می‌دهد. حالا می‌توانید پیش‌بینی انجام دهید و توجه کنید که مدل دو برچسب خروجی می‌دهد:

```
>>> knn_clf.predict([some_digit])
array([[False,  True]])
```

© 2025 Mohsen Zare. All rights reserved.

و جواب درست می‌دهد! عدد ۵ واقعاً بزرگ نیست (False) و فرد است (True).

راه‌های مختلفی برای ارزیابی یک طبقه‌بند چندبرچسبی وجود دارد و انتخاب معیار مناسب بستگی به پروژه شما دارد. یک روش این است که F_1 score را برای هر برچسب جداگانه بسنجید (یا هر معیار دیگری که برای طبقه‌بندهای باینری پیش‌تر بحث شد)، سپس میانگین این امتیازها را محاسبه کنید. کد زیر میانگین F_1 score را در تمام برچسب‌ها محاسبه می‌کند:

```
y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
f1_score(y_multilabel, y_train_knn_pred, average="macro")
0.976410265560605
```

این روش فرض می‌کند که همه برچسب‌ها به یک اندازه اهمیت دارند، که ممکن است همیشه درست نباشد. به‌ویژه اگر تعداد تصاویر آلیس بیشتر از باب یا چارلی باشد، ممکن است بخواهید وزن بیشتری به امتیاز طبقه‌بند روی تصاویر آلیس بدهید. یک راه ساده این است که به هر برچسب وزنی برابر با «حمایت($support$)» آن (یعنی تعداد نمونه‌های دارای آن برچسب) اختصاص دهید. برای این کار کافی است هنگام فراخوانی تابع f1_score() مقدار "average="weighted را تنظیم کنید.

اگر بخواهید از طبقه‌بند ای استفاده کنید که به‌صورت ذاتی از طبقه‌بندی چندبرچسبی پشتیبانی نمی‌کند، مانند SVC، یک راهکار ممکن این است که برای هر برچسب یک مدل جداگانه آموزش دهید. با این حال، این روش ممکن است در درک وابستگی‌های بین برچسب‌ها مشکل داشته باشد. برای مثال، یک رقم بزرگ (۷، ۸ یا ۹) دو برابر احتمال دارد که فرد باشد تا زوج، اما مدل مربوط به برچسب «فرد» نمی‌داند مدل برچسب «بزرگ» چه پیش‌بینی‌ای کرده است. برای حل این مشکل، می‌توان مدل‌ها را به صورت زنجیره‌ای سازماندهی کرد: وقتی یک مدل پیش‌بینی انجام می‌دهد، علاوه بر ویژگی‌های ورودی، از همه پیش‌بینی‌های مدل‌های قبلی در زنجیره نیز استفاده می‌کند.

خبر خوب این است که Scikit-Learn کلاسی به نام ChainClassifier دارد که دقیقاً همین کار را انجام می‌دهد! به‌طور پیش‌فرض، این کلاس برای آموزش از برچسب‌های واقعی استفاده می‌کند و به هر مدل برچسب‌های مناسب را بسته به جایگاه آن در زنجیره می‌دهد. اما اگر پارامتر cv را تنظیم کنید، از روش اعتبارسنجی متقاطع استفاده می‌کند تا پیش‌بینی‌های «تمیز» (خارج از نمونه) هر مدل آموزش‌دیده را برای هر نمونه در مجموعه آموزش به‌دست آورد، و سپس این پیش‌بینی‌ها برای آموزش همه مدل‌ها در ادامه زنجیره استفاده می‌شوند. در ادامه مثالی آورده شده که نشان می‌دهد چطور

© 2025 Mohsen Zare. All rights reserved.

می‌توان با استفاده از استراتژی اعتبارسنجی متقاطع، یک ChainClassifier ساخت و آموزش داد. برای سرعت بخشیدن، فقط از ۲,۰۰۰ تصویر اول در مجموعه آموزش استفاده خواهیم کرد.

```
from sklearn.multioutput import ClassifierChain

chain_clf = ClassifierChain(SVC(), cv=3, random_state=42)
chain_clf.fit(X_train[:2000], y_multilabel[:2000])
```

حالا می‌توانیم از این ChainClassifier برای انجام پیش‌بینی استفاده کنیم:

```
>>> chain_clf.predict([some_digit])
array([[0., 1.]])
```

© 2025 Mohsen Zare. All rights reserved.

طبقه‌بندی چندخروجی

آخرین نوع مسئله طبقه‌بندی که اینجا بررسی می‌کنیم، «طبقه‌بندی چندخروجی-چندکلاسه (*multioutput multiclass classification*)» یا به‌اختصار «طبقه‌بندی چندخروجی (*multioutput classification*)» است. این نوع طبقه‌بندی توسعه‌ای از طبقه‌بندی چندبرچسبی است که در آن هر برچسب می‌تواند چندکلاسه باشد (یعنی بیش از دو مقدار ممکن داشته باشد).

برای روشن شدن موضوع، بیایید سیستمی بسازیم که نویز تصاویر را حذف کند. این سیستم به‌عنوان ورودی، یک تصویر رقم نویزی دریافت می‌کند و (امیدواریم) تصویری پاک‌شده و بدون نویز را خروجی می‌دهد که به‌صورت آرایه‌ای از شدت پیکسل‌ها نمایش داده می‌شود، درست مانند تصاویر MNIST. توجه کنید که خروجی این طبقه‌بندی چندبرچسبی است (هر پیکسل یک برچسب دارد) و هر برچسب می‌تواند مقادیر مختلفی داشته باشد (شدت پیکسل‌ها از ۰ تا ۲۵۵ متغیر است). بنابراین این یک نمونه از سیستم طبقه‌بندی چندخروجی است.

> **نکته**
>
> مرز بین طبقه‌بندی و رگرسیون گاهی مبهم است، همان‌طور که در این مثال دیده می‌شود. می‌توان گفت پیش‌بینی شدت پیکسل بیشتر شبیه به رگرسیون است تا طبقه‌بندی. علاوه بر این، سیستم‌های چندخروجی محدود به وظایف طبقه‌بندی نیستند؛ حتی می‌توانید سیستمی داشته باشید که برای هر نمونه چند برچسبِ خروجی تولید کند، شامل هم برچسب‌های کلاسی و هم برچسب‌های مقداری.

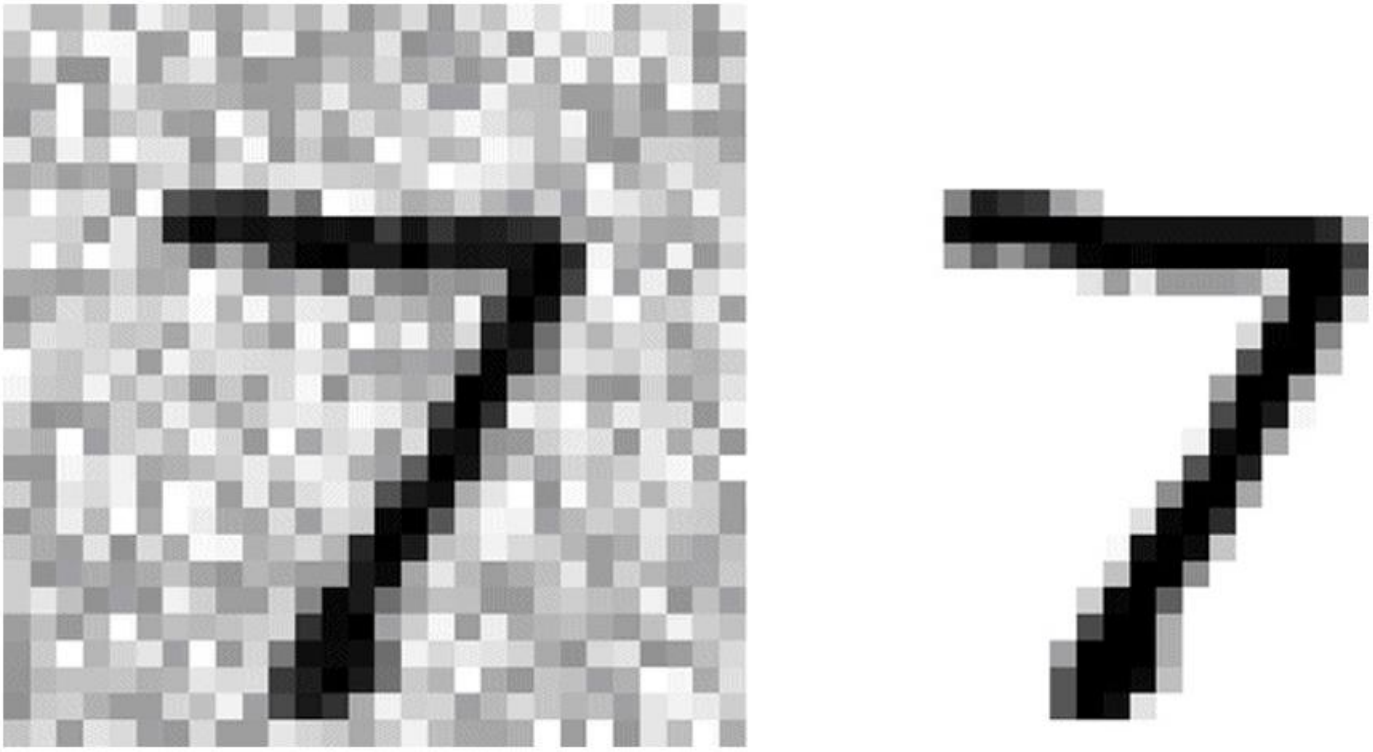
بیایید با ایجاد مجموعه‌های آموزش و آزمون شروع کنیم. برای این کار، تصاویر MNIST را می‌گیریم و با استفاده از تابع randint() از کتابخانه NumPy به شدت پیکسل‌های آن‌ها نویز اضافه می‌کنیم. تصاویر هدف همان تصاویر اصلی خواهند بود:

```
np.random.seed(42)  # to make this code example reproducible
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
```

© 2025 Mohsen Zare. All rights reserved.

```
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test
```

بیایید نگاهی به اولین تصویر از مجموعه‌ی آزمون بیندازیم (شکل ۳-۱۲). بله، ما داریم به داده‌های آزمون سرک می‌کشیم، پس الان باید اخم کرده باشید.

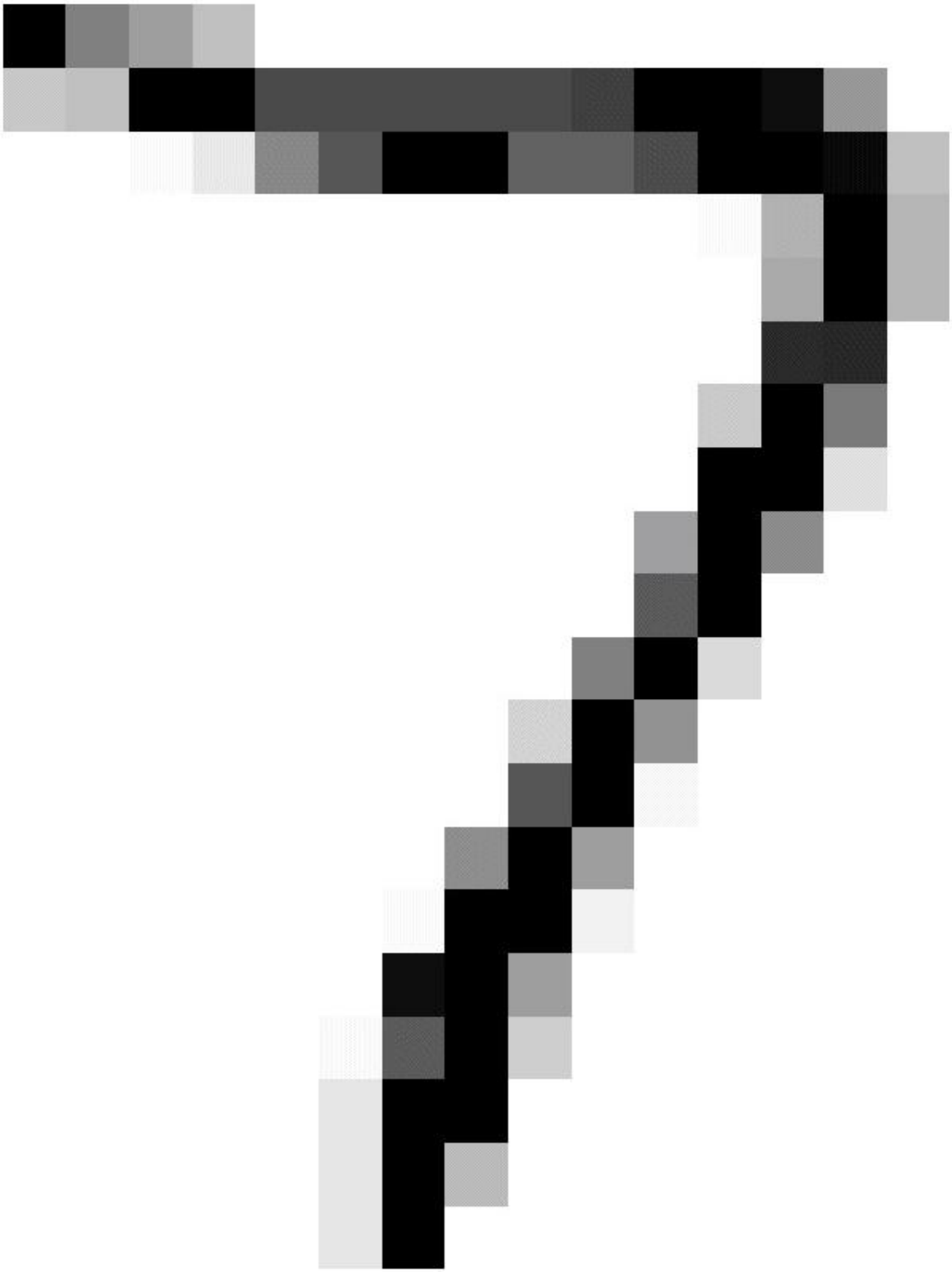


شکل ۳-۱۲. یک تصویر نویزی (چپ) و تصویر هدف بدون نویز (راست)

در سمت چپ تصویر ورودی نویزی قرار دارد، و در سمت راست تصویر تمیز هدف را می‌بینید. حالا بیایید طبقه‌بند را آموزش دهیم و از آن بخواهیم این تصویر را پاک‌سازی کند (شکل ۳-۱۳):

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[0]])
plot_digit(clean_digit)
plt.show()
```

© 2025 Mohsen Zare. All rights reserved.



شکل ۳-۱۳. تصویر پاک‌سازی‌شده

© 2025 Mohsen Zare. All rights reserved.

به نظر می‌رسد نتیجه تقریباً شبیه به تصویر هدف است! با این بخش، مرور ما بر طبقه‌بندی به پایان می‌رسد.

حالا می‌دانید که چگونه معیارهای مناسب برای مسائل طبقه‌بندی را انتخاب کنید، موازنه‌ی میان دقت و recall را تنظیم کنید، طبقه‌بندها را با هم مقایسه کنید و به طور کلی سیستم‌های طبقه‌بندی خوبی برای انواع مسئله ها بسازید. در فصل‌های بعدی، یاد خواهید گرفت که تمام این مدل‌های یادگیری ماشینی که تا اینجا استفاده کرده‌اید، در واقع چگونه کار می‌کنند.

© 2025 Mohsen Zare. All rights reserved.

تمرین‌ها

۱. سعی کنید یک طبقه‌بند برای مجموعه داده MNIST بسازید که روی مجموعه‌ی آزمون به دقت بالای ۹۷% برسد. راهنمایی: طبقه‌بند KNeighborsClassifier برای این کار به‌خوبی عمل می‌کند؛ فقط لازم است مقادیر مناسب برای هایپرپارامترها پیدا کنید (یک جستجوی شبکه‌ای روی هایپرپارامترهای weights و n_neighbors امتحان کنید).

۲. یک تابع بنویسید که بتواند تصویر MNIST را به هر جهتی (چپ، راست، بالا یا پایین) به اندازه‌ی یک پیکسل جابجا کند. سپس برای هر تصویر در مجموعه‌ی آموزش، چهار نسخه‌ی جابجا شده (یکی برای هر جهت) ایجاد کرده و به مجموعه‌ی آموزش اضافه کنید. در نهایت، بهترین مدل خود را روی این مجموعه‌ی آموزشِ گسترش یافته آموزش داده و دقت آن را روی مجموعه‌ی آزمون بسنجید. مشاهده خواهید کرد که مدل شما اکنون عملکرد بهتری دارد! این تکنیک گسترش مصنوعی مجموعه‌ی آموزش، «تقویت داده» (*data augmentation*) یا «گسترش مجموعه‌ی آموزش» (*training set expansion*) نامیده می‌شود.

۳. با داده‌های مجموعه Titanic کار کنید. یک نقطه شروع عالی، سایت Kaggle است. یا می‌توانید داده‌ها را از این لینک *https://homl.info/titanic.tgz* دانلود کرده و مانند داده‌های مسکن در فصل ۲، این فایل tar را از حالت فشرده خارج کنید. این کار دو فایل CSV به شما می‌دهد: train.csv و test.csv که می‌توانید با استفاده از pandas.read_csv() آن‌ها را بارگذاری کنید. هدف این است که مدلی آموزش دهید که بتواند ستون «Survived» را بر اساس سایر ستون‌ها پیش‌بینی کند.

۴. یک طبقه‌بندِ هرزنامهٔ (spam) بسازید (تمرینی چالش‌برانگیزتر):

الف. نمونه‌هایی از هرزنامه‌ها (spam) و نامه‌های معمولی (ham) را از مجموعه داده‌های عمومی Apache SpamAssassin دانلود کنید.

ب. مجموعه داده را از حالت فشرده خارج کنید و با فرمت داده‌ها آشنا شوید.

© 2025 Mohsen Zare. All rights reserved.

ج. داده‌ها را به مجموعه آموزش و مجموعه آزمون تقسیم کنید.

د. یک خط لولهٔ پردازشِ آماده‌سازی داده بنویسید که هر ایمیل را به یک بردار ویژگی تبدیل کند. خط لولهٔ پردازشِ آماده‌سازی شما باید ایمیل را به برداری (پراکنده) تبدیل کند که حضور یا عدم حضور هر کلمه ممکن را نشان دهد. برای مثال، اگر همه ایمیل‌ها فقط شامل چهار کلمه ««Hello»، «how»، «are»» و «you» باشند، ایمیل «Hello you Hello Hello you» به بردار [1, 0, 0, 1] تبدیل می‌شود (به معنی اینکه «Hello» حضور دارد، «how» نیست، «are» نیست و «you» حضور دارد) یا اگر بخواهید تعداد تکرار هر کلمه را بشمارید، به [3, 0, 0, 2]تبدیل می‌شود.
ممکن است بخواهید هایپرپارامترهایی به خط لولهٔ پردازشِ آماده‌سازی خود اضافه کنید تا کنترل کنید آیا سربرگ‌های ایمیل(headers) حذف شوند، هر ایمیل به حروف کوچک تبدیل شود، علائم نگارشی حذف شوند، تمام URLها با کلمهٔ «URL» جایگزین شوند، تمام اعداد با «NUMBER» جایگزین شوند، یا حتی «عملیات ریشه‌یابی» (*stemming*) انجام شود (یعنی انتهای کلمات کوتاه شود؛ کتابخانه‌های پایتونی برای این کار موجود است).

ه. در نهایت، چندین مدل طبقه‌بندی را امتحان کنید و ببینید آیا می‌توانید یک طبقه‌بند هرزنامه عالی بسازید که هم recall (recall) و هم دقت (precision) بالایی داشته باشد.

© 2025 Mohsen Zare. All rights reserved.

فصل ۴. مدل‌های آموزشی

تا اینجا، ما با مدل‌ها و الگوریتم‌های یادگیری ماشین عمدتاً به‌عنوان جعبه‌های سیاه برخورد کرده‌ایم. اگر تمرین‌های فصل‌های پیشین را انجام داده باشید، احتمالاً تعجب کرده‌اید که بدون دانستن جزئیات پشت صحنه، چه کارهای زیادی می‌توان انجام داد: یک سیستم رگرسیون را بهینه کرده‌اید، یک طبقه‌بندی‌کننده اعداد دست‌نویس را بهبود داده‌اید و حتی یک طبقه‌بندی‌کننده هرزنامه را از صفر ساخته‌اید—همه این‌ها بدون اینکه واقعاً بدانید الگوریتم‌ها چگونه عمل می‌کنند. در واقع، در بسیاری از موقعیت‌ها، دانستن جزئیات پیاده‌سازی کاملاً ضروری نیست.

با این حال، داشتن درک عمیق از عملکرد مدل‌ها به شما این امکان را می‌دهد که سریع‌تر مدل مناسب، الگوریتم آموزشی متناسب و مجموعه‌ای بهینه از هایپرپارامترها را انتخاب کنید. دانستن آنچه در پشت پرده رخ می‌دهد، همچنین به شما کمک می‌کند مشکلات را سریع‌تر شناسایی و برطرف کنید و تجزیه و تحلیل خطاها را با دقت بیشتری انجام دهید. علاوه بر این، بسیاری از مفاهیم مطرح‌شده در این فصل، پایه‌ای ضروری برای درک، ساخت و آموزش شبکه‌های عصبی هستند که در بخش دوم کتاب به تفصیل بررسی خواهند شد.

در این فصل، کار خود را با مدل رگرسیون خطی آغاز می‌کنیم؛ یکی از ساده‌ترین مدل‌های یادگیری ماشین. ما دو روش کاملاً متفاوت برای آموزش این مدل بررسی خواهیم کرد:

© 2025 Mohsen Zare. All rights reserved.

- روش «راه‌حل مستقیم مدل» یا معادله‌ی «فرم بسته (Closed-Form)» : در این روش، پارامترهای مدل به صورت مستقیم و با استفاده از یک معادله تحلیلی محاسبه می‌شوند، به‌طوری که بهترین برازش مدل با مجموعه آموزشی به دست آید؛ به عبارت دیگر، پارامترهایی که تابع هزینه را روی داده‌های آموزشی مینیمم می‌کنند.
- گرادیان نزولی (Gradient Descent): این رویکرد بهینه‌سازی تکراری، پارامترهای مدل را به تدریج و گام‌به‌گام تغییر می‌دهد تا تابع هزینه روی مجموعه آموزشی مینیمم شود و در نهایت به همان مقادیر پارامترهایی برسد که در روش «فرم بسته» به دست آمده‌اند. در ادامه، چند نوع گرادیان نزولی را بررسی خواهیم کرد که در بخش دوم کتاب و هنگام آموزش شبکه‌های عصبی بارها مورد استفاده قرار خواهند گرفت: گرادیان نزولی دسته‌ای (Batch GD)، گرادیان نزولی مینی دسته‌ای (Mini-Batch GD) و گرادیان نزولی تصادفی (Stochastic GD).

سپس به «رگرسیون چندجمله‌ای» می‌پردازیم؛ مدلی پیچیده‌تر که قادر است داده‌های غیرخطی را نیز برازش دهد. از آنجا که این مدل تعداد پارامترهای بیشتری نسبت به رگرسیون خطی دارد، احتمال بیش‌برازش روی داده‌های آموزشی در آن بالاتر است. در ادامه، روش استفاده از منحنی‌های یادگیری برای تشخیص بیش‌برازش را بررسی خواهیم کرد و سپس با چند تکنیک «منظم‌سازی» می‌شویم که می‌توانند ریسک بیش‌برازش را کاهش دهند.

در پایان، به بررسی دو مدل دیگر می‌پردازیم که معمولاً در مسائل طبقه‌بندی کاربرد دارند: «رگرسیون لُجِستیک» و «رگرسیون سافت‌مَکس».

هشدار

در این فصل با تعداد زیادی معادله ریاضی روبرو خواهید شد که بر مفاهیم پایه جبر خطی و حساب دیفرانسیل و انتگرال متکی هستند. برای درک این معادلات، باید با موارد زیر آشنا باشید: بردارها و ماتریس‌ها، نحوه ترانهاده کردن، ضرب و معکوس کردن آنها و مفهوم مشتقات جزئی. اگر با این مفاهیم آشنایی ندارید، توصیه می‌کنیم ابتدا آموزش‌های مقدماتی جبر خطی و حساب دیفرانسیل و انتگرال را که به صورت نوت‌بوک‌های Jupyter در مطالب تکمیلی آنلاین موجود است، مطالعه کنید. برای کسانی که حساسیت چندانی نسبت به ریاضیات ندارند، مطالعه این فصل همچنان مفید است: می‌توان از معادلات صرف‌نظر کرد و روی متن اصلی تمرکز کرد، زیرا متن به اندازه کافی توضیح داده شده تا مفاهیم کلیدی را درک کنید.

© 2025 Mohsen Zare. All rights reserved.

رگرسیون خطی

در فصل اول، با یک مدل رگرسیون ساده برای پیش‌بینی رضایت از زندگی آشنا شدیم:

$$life_satisfaction = \theta_0 + \theta_1 \times GDP_per_capita$$

این مدل صرفاً یک تابع خطی از ویژگی ورودی GDP_per_capita (سرانه تولید ناخالص داخلی) است. θ_0و θ_1 پارامترهای مدل محسوب می‌شوند.

به طور کلی، یک مدل خطی پیش‌بینی‌های خود را با محاسبه‌ی مجموع وزنی ویژگی‌های ورودی و افزودن یک عبارت ثابت به نام «ترم بایاس ($bias\ term$)» انجام می‌دهد که گاهی به آن «ترم ثابت (عرض از مبدأ) ($intercept\ term$)» نیز گفته می‌شود، همان‌طور که در معادله‌ی ۴-۱ مشاهده می‌کنید.

معادله ۴-۱. پیش‌بینی مدل رگرسیون خطی

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

در این معادله:

- $\hat{y}$ نشان‌دهنده‌ی «مقدار پیش‌بینی‌شده» است.
- n تعداد «ویژگی‌ها» را مشخص می‌کند.
- x_i مقدار «ویژگی i اُم» است.
- θ_j پارامتر j اُم مدل است که شامل «عبارت بایاس θ_0 » و «وزن‌های ویژگی‌ها θ_1،θ_2، $\cdots$، θ_n» می‌باشد.

این معادله را می‌توان به شکل «برداری» بازنویسی کرد تا نوشتار آن بسیار ساده‌تر شود، همان‌طور که در معادله ۴-۲ نشان داده شده است.

© 2025 Mohsen Zare. All rights reserved.

معادله ۴-۲. پیش‌بینی مدل رگرسیون خطی (فرم برداری)

$$\hat{y} = h_\theta(x) = \theta^T \cdot x$$

در این معادله:

- $h_{\boldsymbol{\theta}}$ تابع فرضیه است که از پارامترهای مدل $\boldsymbol{\theta}$ استفاده می‌کند.
- $\boldsymbol{\theta}$ بردار پارامترهای مدل است که شامل عبارت بایاس θ_0 و وزن‌های ویژگی θ_1 تا θ_n می‌باشد.
- $\mathbf{x}$ بردار ویژگی‌های نمونه است که شامل x_0 تا x_n است و همیشه $x_0 = 1$ است.
- $\boldsymbol{\theta} \cdot \mathbf{x}$ حاصلضرب داخلی بردارهای $\boldsymbol{\theta}$ و $\mathbf{x}$ است که برابر است با:

$$\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

نکته

در یادگیری ماشین، معمولاً بردارها به صورت «بردارهای ستونی (*column vector*)» نمایش داده می‌شوند، یعنی آرایه‌های دوبعدی با یک ستون. اگر $\boldsymbol{\theta}$ و $\mathbf{x}$ بردارهای ستونی باشند، پیش‌بینی به شکل $\hat{y} = \theta^T x$ نوشته می‌شود، که در آن θ^T ترانهاده(*transpose*) $\boldsymbol{\theta}$ است (برداری سطری به جای برداری ستونی) و $\theta^T x$ ضرب ماتریسی بین θ^T و $\mathbf{x}$ است. این همان پیش‌بینی اسکالر است، با این تفاوت که اکنون به صورت یک ماتریس تک‌سلولی نمایش داده می‌شود. در این کتاب، از این نمادگذاری استفاده می‌کنیم تا از جابجایی‌های ناخواسته بین ضرب نقطه‌ای و ضرب ماتریسی جلوگیری شود و نگارش معادلات یکدست و واضح باقی بماند.

خب، این همان مدل رگرسیون خطی است—اما چگونه می‌توان آن را آموزش داد؟ به یاد داشته باشید که آموزش مدل یعنی تنظیم پارامترهای آن به گونه‌ای که بهترین برازش را روی داده‌های آموزشی ارائه دهد. برای این کار، ابتدا نیاز به یک معیار داریم که نشان دهد مدل تا چه حد با داده‌ها سازگار است. همان‌طور که در فصل ۲ دیدیم، رایج‌ترین معیار برای سنجش عملکرد مدل‌های رگرسیون، جذر میانگین مربعات خطا (RMSE) است (معادله ۲-۱). بنابراین، بنابراین، آموزش یک مدل رگرسیون خطی به معنای یافتن مقادیر $\boldsymbol{\theta}$ است که RMSE را مینیمم کنند. در عمل، مینیمم کردن میانگین مربعات خطا (MSE)

© 2025 Mohsen Zare. All rights reserved.

ساده‌تر است و همان نتیجه را به دست می‌دهد، زیرا هر مقداری که تابعی مثبت را مینیمم کند، جذر آن نیز به مینیمم خواهد رسید.

هشدار

در عمل، الگوریتم‌های یادگیری معمولاً در طول آموزش، تابع زیانی متفاوت از معیار عملکرد نهایی که برای ارزیابی مدل استفاده می‌شود، بهینه می‌کنند. دلیل این امر اغلب این است که بهینه‌سازی آن تابع خاص آسان‌تر است و یا نیاز به عبارات اضافی دارد، مانند مواردی که برای منظم‌سازی اضافه می‌شوند. یک معیار عملکرد مناسب باید تا حد امکان با هدف نهایی کسب‌وکار همسو باشد، در حالی که یک تابع زیان خوب باید به راحتی قابل بهینه‌سازی باشد و همبستگی قوی با معیار عملکرد داشته باشد. به عنوان مثال، طبقه‌بندی‌کننده‌ها اغلب با استفاده از تابع هزینه‌ای مانند log loss آموزش داده می‌شوند، در حالی که ارزیابی نهایی آن‌ها با معیارهایی مانند precision یا recall انجام می‌شود. کمینه کردن log loss نسبتاً ساده است و انجام آن معمولاً باعث بهبود precision یا recall مدل نیز می‌شود.

مقدار MSE برای فرضیه رگرسیون خطی $h_{\boldsymbol{\theta}}$ روی مجموعه آموزشی $\mathbf{X}$ را می‌توان با معادله ۴-۳ محاسبه کرد.

معادله ۴-۳. تابع هزینه MSE برای یک مدل رگرسیون خطی

$$\mathrm{MSE}(\mathrm{X}, h_{\theta}) = 1m \sum_{i=1}^{m} \left(\theta^T x^{(i)} - y^{(i)}\right)^2$$

بیشتر این نمادها در فصل ۲ معرفی شده‌اند (به بخش «نمادگذاری‌ها» مراجعه کنید). تنها تفاوت این است که برای مشخص کردن اینکه مدل با بردار پارامتر $\boldsymbol{\theta}$ تعریف شده است، به جای h، از $h_{\boldsymbol{\theta}}$ استفاده می‌کنیم. برای ساده‌تر کردن نمادگذاری، به جای $\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}})$ تنها $\mathrm{MSE}(\boldsymbol{\theta})$ را می‌نویسیم.

© 2025 Mohsen Zare. All rights reserved.

معادله نرمال

برای پیدا کردن بردار $\boldsymbol{\theta}$ که MSE را به حداقل می‌رساند، می‌توان از یک «راه‌حل فرم بسته (*closed-form solution*)» استفاده کرد؛ به این معنا که یک معادله ریاضی داریم که نتیجه را مستقیماً ارائه می‌دهد. این معادله را «معادله نرمال (*Normal equation*)» می‌نامند (معادله ۴-۴).

معادله ۴-۴. معادله نرمال

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

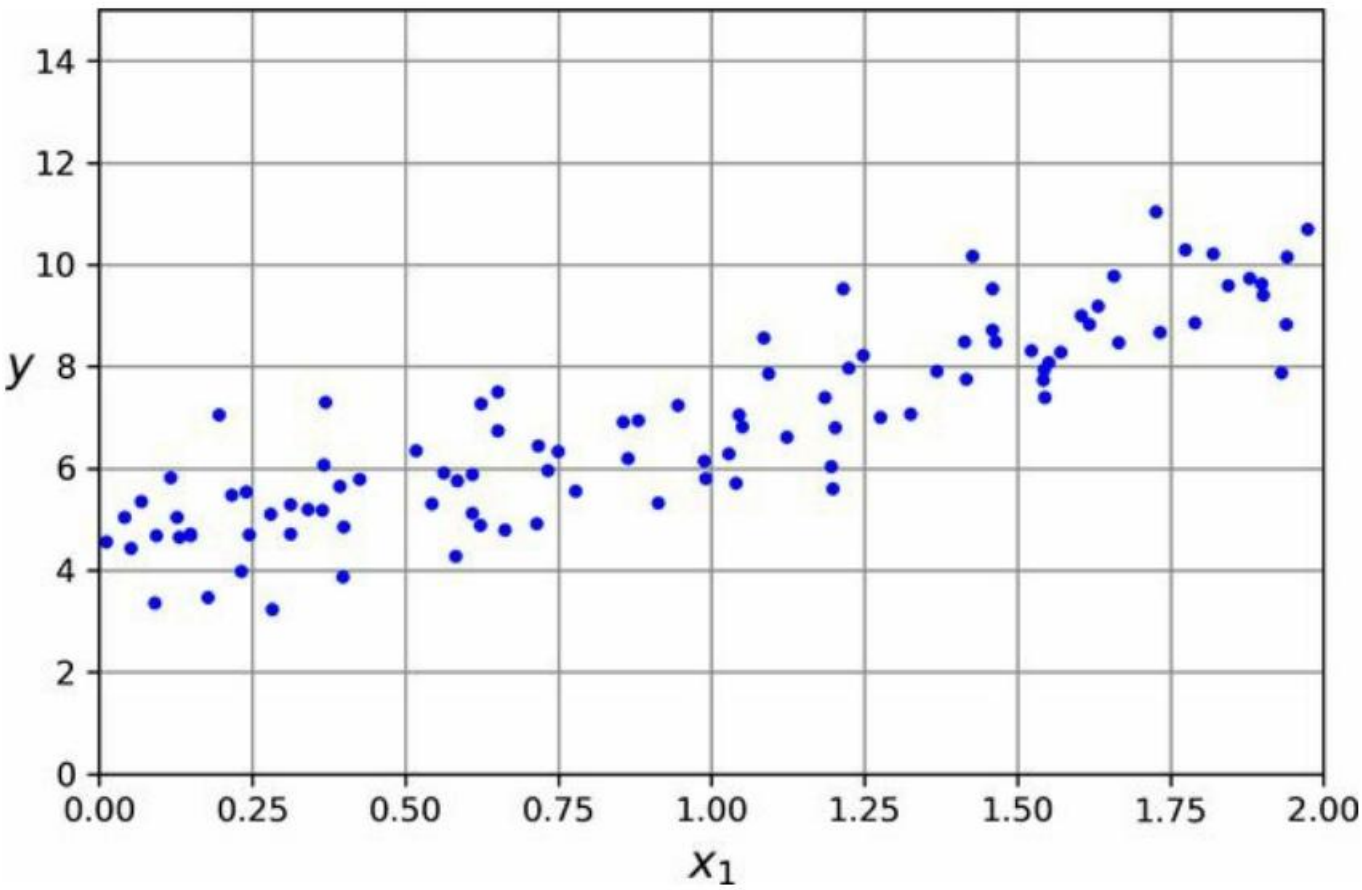
در این معادله:

- $\hat{\theta}$ مقداری از $\boldsymbol{\theta}$ است که تابع هزینه را مینیمم می‌کند(به حداقل میرساند).
- $\mathbf{y}$ بردار مقادیر هدف است که شامل $y^{(1)}$ تا $y^{(m)}$ می‌باشد.

حال بیایید یک مجموعه داده با الگوی خطی برای آزمایش این معادله تولید کنیم (شکل ۴-۱):

```
import numpy as np

np.random.seed(42) # to make this code example reproducible
m = 100 # number of instances
X = 2 * np.random.rand(m, 1) # column vector
y = 4 + 3 * X + np.random.randn(m, 1) # column vector
```

© 2025 Mohsen Zare. All rights reserved.



شکل ۴-۱. یک مجموعه داده خطی تصادفی تولید شده

حالا می‌خواهیم $\hat{\theta}$ را با استفاده از معادله نرمال محاسبه کنیم. در اینجا از تابع inv() در ماژول جبر خطی (np.linalg) از کتابخانه‌ی NumPy برای محاسبه‌ی معکوس ماتریس و از متد dot() برای ضرب ماتریس‌ها استفاده می‌کنیم:

```
from sklearn.preprocessing import add_dummy_feature

X_b = add_dummy_feature(X) # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

© 2025 Mohsen Zare. All rights reserved.

نکته

عملگر @ برای انجام ضرب ماتریسی استفاده می‌شود. اگر A و B آرایه‌های NumPy باشند، عبارت A@B معادل np.matmul(A, B) است. بسیاری از کتابخانه‌های دیگر مانند TensorFlow، PyTorch و JAX نیز از عملگر @ پشتیبانی می‌کنند. با این حال، استفاده از @ در آرایه‌های پایتون خالص (یعنی لیست‌هایی از لیست‌ها) امکان‌پذیر نیست.

تابعی که برای تولید داده‌ها استفاده کردیم به صورت $y = 4 + 3x_1 + \text{Gaussiannoise}$ است. بیایید بررسی کنیم این معادله چه مفهومی را نشان می‌دهد:

```
>>> theta_best
array([[4.21509616],
[2.77011339]])
```

ما به جای مقادیر واقعی $\theta_0 = 4.215$ و $\theta_1 = 2.770$ انتظار داشتیم $\theta_0 = 4$ و $\theta_1 = 3$ باشد. این مقادیر به اندازه کافی نزدیک بودند، اما وجود نویز، recall دقیق پارامترهای تابع اصلی را غیرممکن می‌کرد. هرچه مجموعه داده‌ها کوچک‌تر و نویزدارتر باشد، تخمین دقیق پارامترها دشوارتر می‌شود.

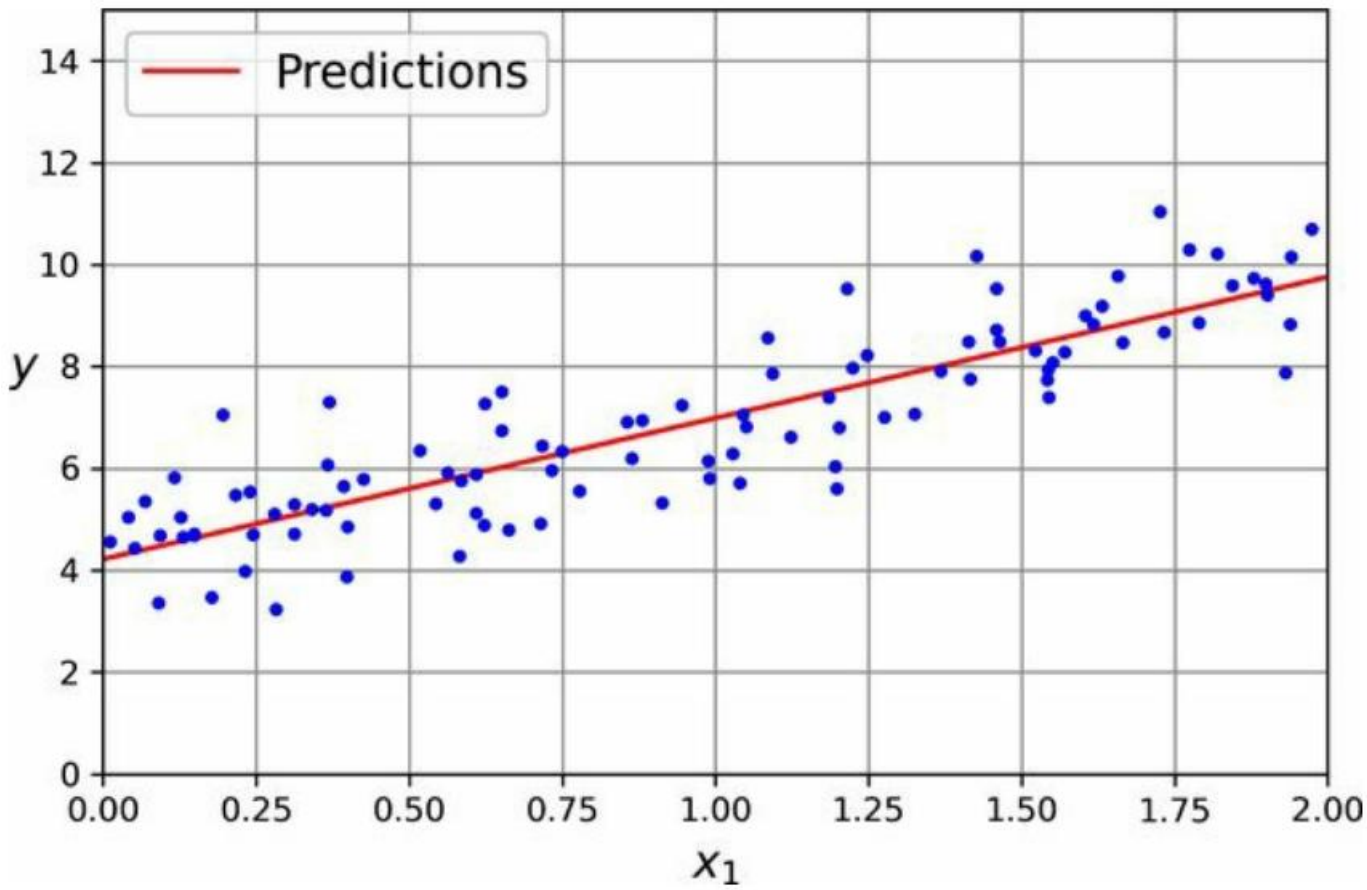
اکنون می‌توانیم با استفاده از $\hat{\theta}$ پیش‌بینی‌ها را انجام دهیم:

```
>>> X_new = np.array([[0], [2]])
>>> X_new_b = add_dummy_feature(X_new) # add x0 = 1 to each instance
>>> y_predict = X_new_b @ theta_best
>>> y_predict
array([[4.21509616],
[9.75532293]])
```

© 2025 Mohsen Zare. All rights reserved.

بیایید پیش‌بینی‌های این مدل را رسم کنیم (شکل ۴-۲):

```
import matplotlib.pyplot as plt

plt.plot(X_new, y_predict, "r-", label="Predictions")
plt.plot(X, y, "b.")
[...] # beautify the figure: add labels, axis, grid, and legend
plt.show()
```



شکل ۴-۲. پیش‌بینی‌های مدل رگرسیون خطی

انجام رگرسیون خطی با استفاده از Scikit-Learn نسبتاً ساده است:

© 2025 Mohsen Zare. All rights reserved.

```
>>> from sklearn.linear_model import LinearRegression
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([4.21509616]), array([[2.77011339]]))
>>> lin_reg.predict(X_new)
array([[4.21509616],
[9.75532293]])
```

توجه داشته باشید که Scikit-Learn، ترمِ بایاس (intercept_) را از وزن‌های ویژگی (coef_) جدا می‌کند. کلاس LinearRegression بر پایه تابع scipy.linalg.lstsq() ساخته شده است (که مخفف "least squares" یا حداقل مربعات است) و می‌توانید مستقیماً آن را فراخوانی کنید:

```
>>> theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y, rcond=1e-6)
>>> theta_best_svd
array([[4.21509616],
[2.77011339]])
```

این تابع $\hat{\theta} = X + y$ را محاسبه می‌کند، که در آن $X+$ «شبه‌معکوس (*pseudoinverse*)» $\mathbf{X}$ است (به طور خاص، معکوس مور-پنروز(Moore–Penrose inverse)). می‌توانید از np.linalg.pinv() برای محاسبه مستقیم شبه‌معکوس استفاده کنید:

```
>>> np.linalg.pinv(X_b) @ y
array([[4.21509616],
[2.77011339]])
```

خودِ شبه‌معکوس با استفاده از تکنیک استاندارد فاکتورگیری ماتریس به نام «تجزیه مقدار تکین (SVD: *singular value decomposition*)» محاسبه می‌شود. این روش ماتریس مجموعه آموزشی $\mathbf{X}$ را به حاصل‌ضرب سه ماتریس $\boldsymbol{U\Sigma V}^{\top}$ تجزیه کند (برای جزئیات به numpy.linalg.svd() مراجعه کنید). شبه معکوس به صورت $X+= V\Sigma + U^{\top}$ محاسبه می‌شود. برای به‌دست آوردن Σ، الگوریتم

© 2025 Mohsen Zare. All rights reserved.

ابتدا مقادیر کوچک‌تر از یک آستانه بسیار کوچک را صفر می‌کند، سپس تمام مقادیر غیرصفر را با معکوس آنها جایگزین می‌کند و در نهایت ماتریس حاصل را ترانهاده می‌کند. این روش نسبت به استفاده از معادله نرمال کارآمدتر است و موارد حاشیه‌ای را به خوبی مدیریت می‌کند. در واقع، اگر ماتریس $\mathbf{X}^\top\mathbf{X}$ معکوس‌پذیر نباشد (مثلاً اگر $m < n$ باشد یا برخی ویژگی‌ها اضافی باشند)، معادله نرمال ممکن است کار نکند، اما شبه‌معکوس همیشه تعریف شده است.

© 2025 Mohsen Zare. All rights reserved.

پیچیدگی محاسباتی

معادله نرمال، معکوس $\mathbf{X}^\top\mathbf{X}$ را محاسبه می‌کند که یک ماتریس $(n+1)\times(n+1)$ است که در آن n تعداد ویژگی‌هاست. پیچیدگی محاسباتی (computational complexity) معکوس کردن چنین ماتریسی معمولاً بسته به پیاده‌سازی، بین $O(n^{2.4})$ تا $O(n^3)$ است. به عبارت دیگر، اگر تعداد ویژگی‌ها دو برابر شود، زمان محاسبه تقریباً $2^{2.4} = 5.3$ تا $2^3 = 8$ برابر می‌شود.

روش SVD که توسط کلاس LinearRegression در Scikit-Learn استفاده می‌شود، دارای پیچیدگی تقریباً $O(n^2)$ است. بنابراین، اگر تعداد ویژگی‌ها دو برابر شود، زمان محاسبه تقریباً چهار برابر می‌شود.

هشدار

هر دو روش، معادله نرمال و روش SVD، زمانی که تعداد ویژگی‌ها زیاد باشد (مثلاً ۱۰۰,۰۰۰)، بسیار کند می‌شوند. از سوی دیگر، هر دو روش نسبت به تعداد نمونه‌ها در مجموعه آموزشی پیچیدگی خطی دارند($O(m)$)، و بنابراین می‌توانند مجموعه‌های آموزشی بزرگ را به طور مؤثر مدیریت کنند، مشروط بر اینکه داده‌ها در حافظه جای گیرند.

همچنین، پس از آموزش مدل رگرسیون خطی (با استفاده از معادله نرمال یا هر الگوریتم دیگری)، پیش‌بینی‌ها بسیار سریع انجام می‌شوند. پیچیدگی محاسباتی پیش‌بینی، هم با توجه به تعداد نمونه‌ها و هم تعداد ویژگی‌ها، خطی است. به عبارت دیگر، انجام پیش‌بینی روی دو برابر نمونه‌ها یا دو برابر ویژگی‌ها تقریباً دو برابر زمان می‌برد.

اکنون به روشی کاملاً متفاوت برای آموزش مدل رگرسیون خطی می‌پردازیم، که برای مواقعی که تعداد زیادی ویژگی یا نمونه‌های آموزشی بزرگ وجود دارد و جا دادن آنها در حافظه دشوار است، مناسب‌تر است.

© 2025 Mohsen Zare. All rights reserved.

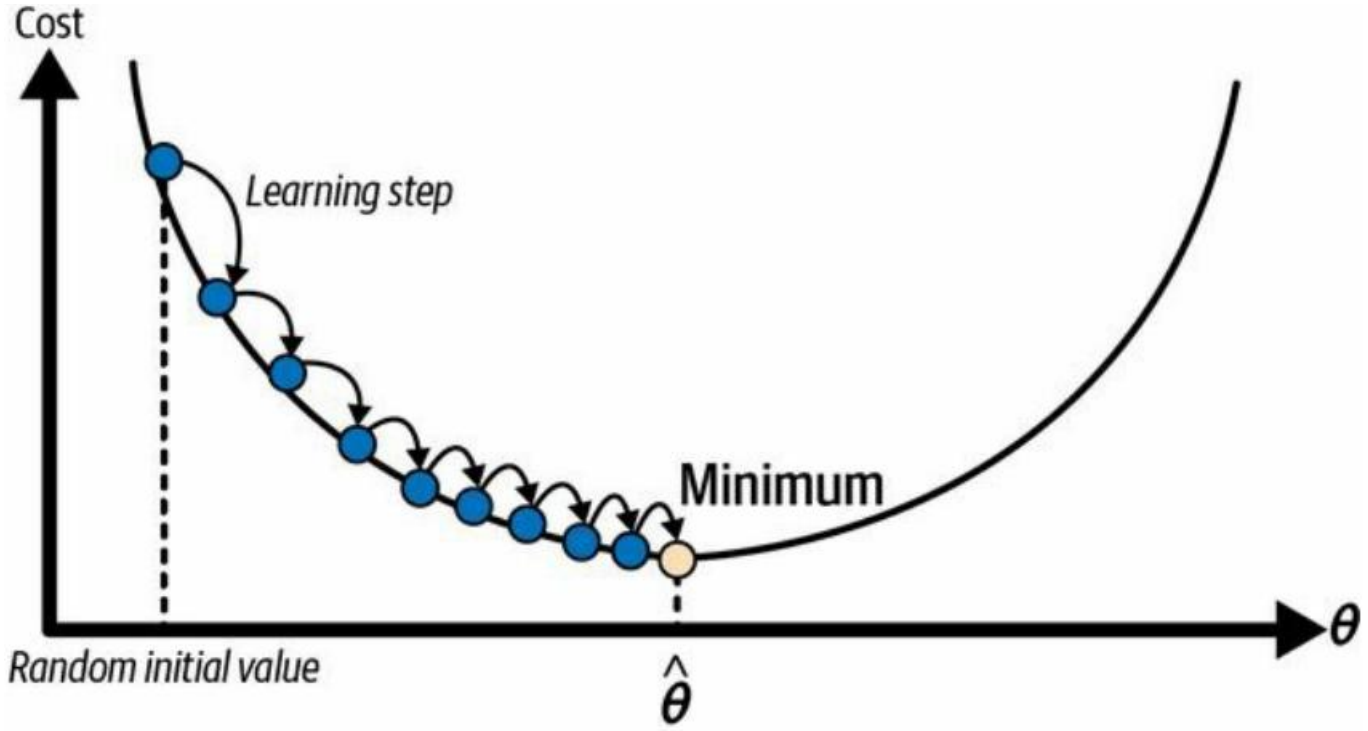
الگوریتم گرادیان نزولی

«گرادیان نزولی (GD: *Gradient Descent*)» یک الگوریتم بهینه‌سازی عمومی است که می‌تواند راه‌حل‌های بهینه برای طیف گسترده‌ای از مسائل پیدا کند. ایده اصلی آن، به‌روزرسانی تکراری پارامترها به‌منظور مینیمم کردن یک تابع هزینه است.

فرض کنید در کوهستان و در مه غلیظ گم شده‌اید و تنها قادر به حس شیب زمین زیر پای خود هستید. یک استراتژی مؤثر برای رسیدن به انتهای دره، حرکت در جهت تندترین شیب است. این همان کاری است که گرادیان نزولی انجام می‌دهد: الگوریتم گرادیان محلی تابع خطا را نسبت به بردار پارامتر $\boldsymbol{\theta}$ محاسبه می‌کند و در جهت منفی گرادیان حرکت می‌کند. زمانی که گرادیان صفر شود، الگوریتم به حداقل تابع هزینه رسیده است.

در عمل، ابتدا پارامترها $\boldsymbol{\theta}$ «با مقادیر تصادفی مقداردهی اولیه (*random initialization*)» می‌شوند. سپس به تدریج بهبود می‌یابند و در هر گام، یک تغییر کوچک انجام می‌شود تا مقدار تابع هزینه، مانند MSE، کاهش یابد. این روند ادامه می‌یابد تا الگوریتم به حداقل همگرا شود (شکل ۴-۳ را ببینید).

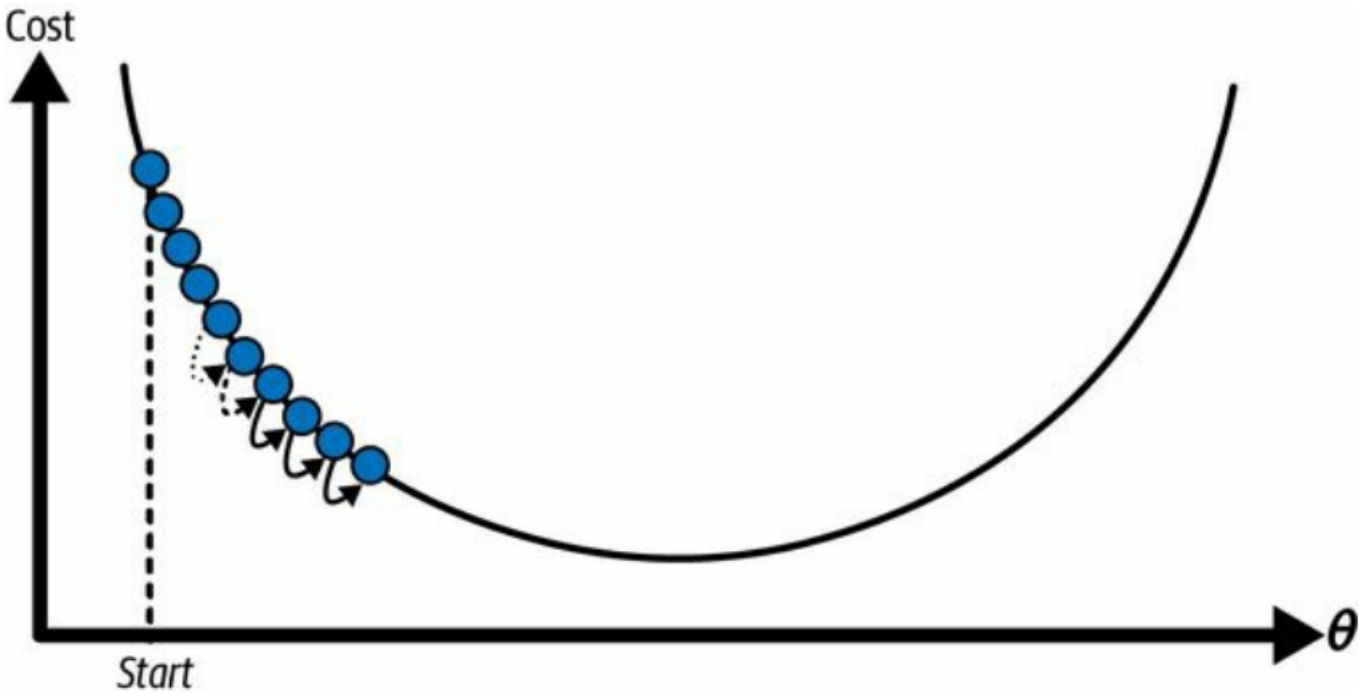
© 2025 Mohsen Zare. All rights reserved.



شکل ۴-۳. در این تصویر مربوط به گرادیان نزولی، پارامترهای مدل با مقادیر تصادفی مقداردهی اولیه می‌شوند و به‌صورت تکراری به‌روزرسانی می‌شوند تا تابع هزینه به حداقل برسد. اندازه گام یادگیری متناسب با شیب تابع هزینه است، بنابراین با نزدیک شدن مقدار تابع هزینه به حداقل، گام‌ها به تدریج کوچک‌تر می‌شوند.

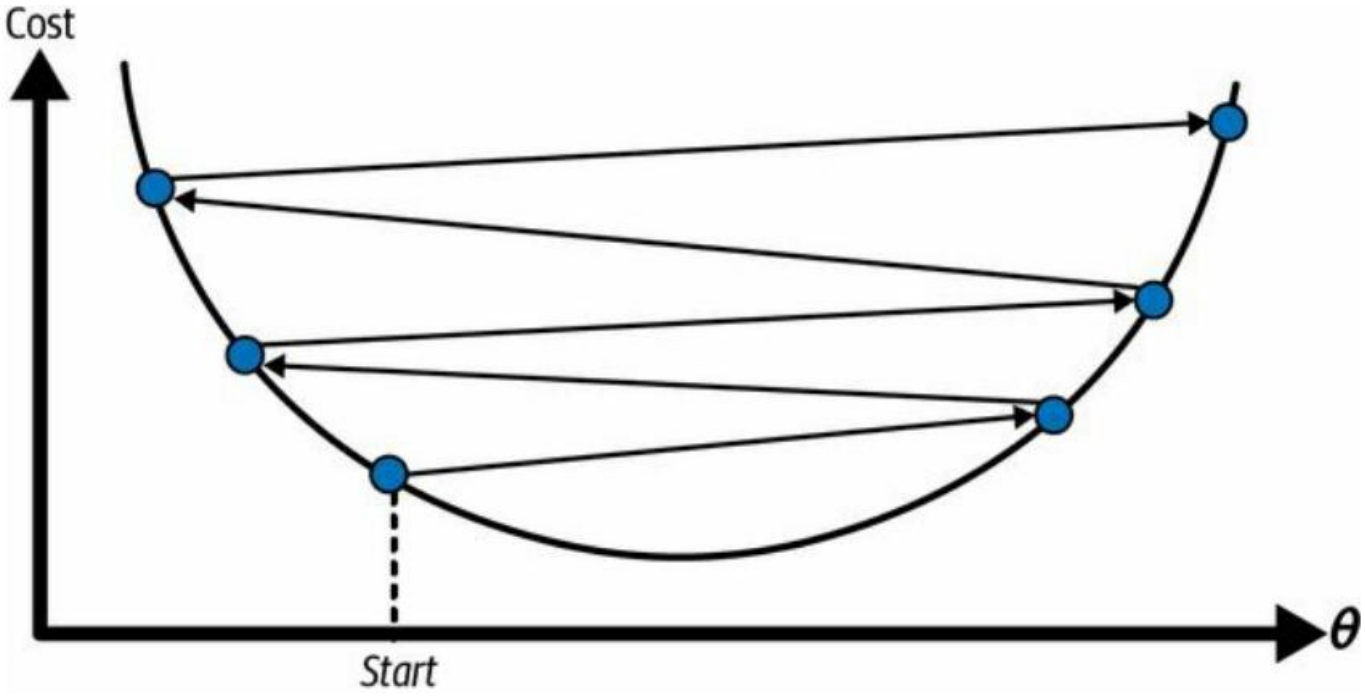
یکی از پارامترهای مهم در گرادیان نزولی، اندازه گام‌ها است که توسط هایپرپارامتر «نرخ یادگیری ($learning\ rate$)» تعیین می‌شود. اگر نرخ یادگیری بسیار کوچک باشد، الگوریتم برای همگرایی به تعداد زیادی تکرار نیاز دارد و این فرایند زمان‌بر خواهد بود (شکل ۴-۴ را ببینید).

© 2025 Mohsen Zare. All rights reserved.



شکل ۴-۴. نرخ یادگیریِ خیلی کوچک

از سوی دیگر، اگر نرخ یادگیری خیلی بزرگ باشد، الگوریتم ممکن است از دره بپرد و به سمت دیگر حرکت کند، گاهی حتی با مقادیر بزرگ‌تر از قبل. این وضعیت می‌تواند باعث «واگرایی الگوریتم» شود و مانع از یافتن یک راه‌حل بهینه گردد (شکل ۴-۵ را ببینید).

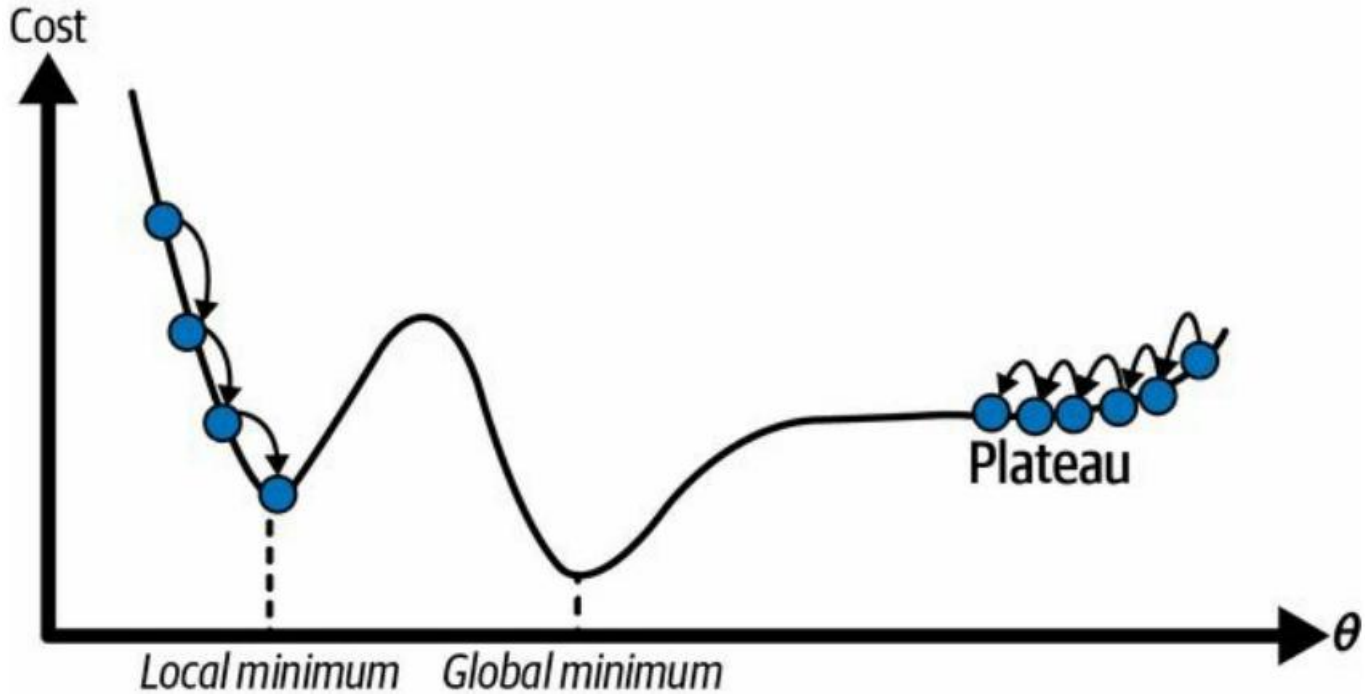


شکل ۴-۵. نرخ یادگیریِ خیلی

© 2025 Mohsen Zare. All rights reserved.

علاوه بر این، همه توابع هزینه منظم و ساده نیستند. ممکن است چاله‌ها، برآمدگی‌ها، فلات‌ها و انواع دیگر سطوح ناهموار وجود داشته باشد که همگرایی به حداقل را دشوار می‌کند. شکل ۴-۶ دو چالش اصلی گرادیان نزولی را نشان می‌دهد:

- اگر مقداردهی اولیه تصادفی الگوریتم را از سمت چپ آغاز کند، ممکن است به یک «مینیمم محلی (*local minimum*)» همگرا شود که با «مینیمم مطلق (*global minimum*)» برابر نیست.
- اگر الگوریتم از سمت راست شروع شود، عبور از بخش‌های بدون تغییر و فلات‌های نمودار (plateau) زمان زیادی طول خواهد کشید.
- همچنین، اگر الگوریتم خیلی زود متوقف شود، هرگز به مینیمم مطلق نمی‌رسد.



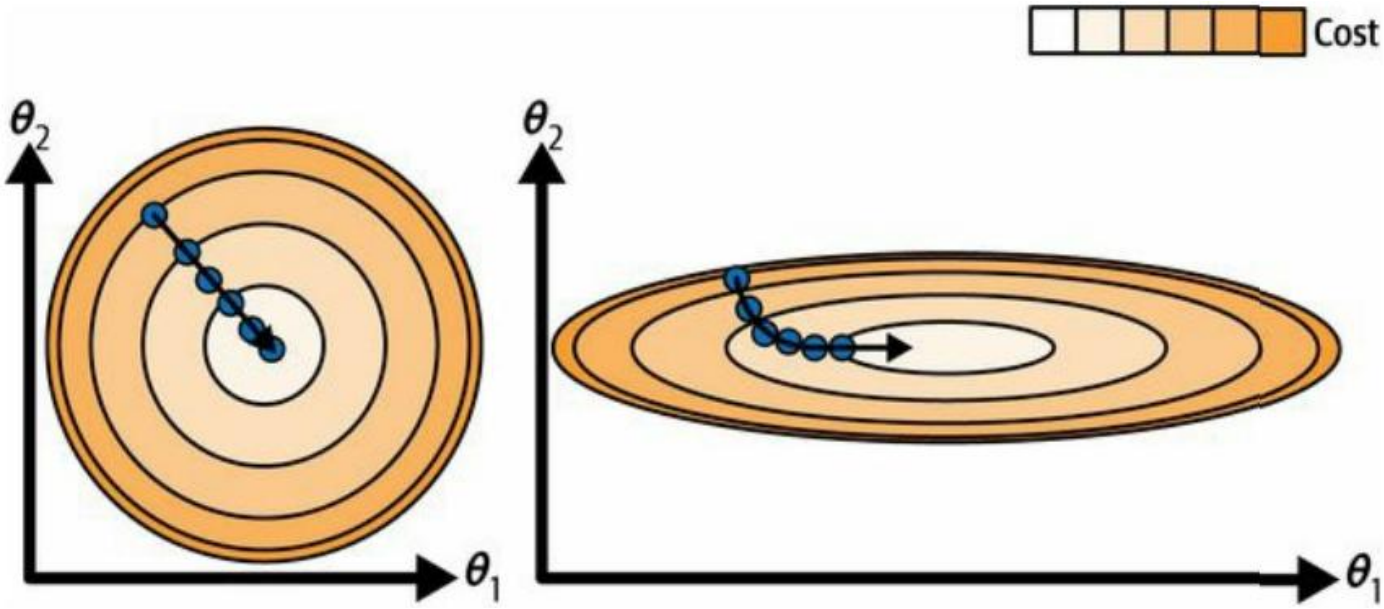
شکل ۴-۶. مشکلات گرادیان نزولی

خوشبختانه، تابع هزینه MSE برای مدل رگرسیون خطی یک «تابع محدب (*convex function*)» است. این بدان معناست که اگر دو نقطه روی منحنی انتخاب کنید، پاره‌خطی که آنها را به هم وصل می‌کند، هرگز زیر منحنی قرار نمی‌گیرد. به عبارت دیگر، هیچ مینیمم محلی وجود ندارد و تنها یک مینیمم مطلق موجود است. علاوه بر این، تابع پیوسته است و شیب آن به طور ناگهانی تغییر نمی‌کند. این

© 2025 Mohsen Zare. All rights reserved.

ویژگی‌ها یک پیامد مهم دارند: گرادیان نزولی تضمین می‌کند که با صبر کافی و انتخاب نرخ یادگیری مناسب، می‌توان به مینیمم مطلق نزدیک شد.

با اینکه تابع هزینه شکل کاسه‌ای دارد، اگر ویژگی‌ها دارای مقیاس‌های بسیار متفاوت باشند، این کاسه می‌تواند کشیده شود. شکل ۴-۷ گرادیان نزولی را روی دو مجموعه آموزشی نشان می‌دهد: در سمت چپ، ویژگی‌های ۱ و ۲ مقیاس یکسانی دارند و در سمت راست، ویژگی ۱ مقادیر بسیار کوچکتری نسبت به ویژگی ۲ دارد.



شکل ۴-۷. گرادیان نزولی با (چپ) و بدون (راست) مقیاس‌بندی ویژگی

همان‌طور که مشاهده می‌کنید، در سمت چپ، الگوریتم گرادیان نزولی به‌طور مستقیم به سمت مینیمم حرکت می‌کند و بنابراین سریعاً به آن می‌رسد. در سمت راست، ابتدا حرکت الگوریتم تقریباً عمود بر جهت مینیمم مطلق است و طی یک مسیر طولانی به سمت پایین دره‌ای نسبتاً مسطح ادامه می‌یابد. در نهایت، الگوریتم به مینیمم خواهد رسید، اما زمان بیشتری طول می‌کشد.

© 2025 Mohsen Zare. All rights reserved.

هشدار

هنگام استفاده از گرادیان نزولی، باید مطمئن شوید که همه ویژگی‌ها دارای مقیاس‌های مشابه هستند (برای مثال با استفاده از کلاس StandardScaler از Scikit-Learn)، در غیر این صورت همگرایی الگوریتم بسیار طولانی‌تر خواهد شد.

این نمودار همچنین نشان می‌دهد که آموزش یک مدل، در واقع جستجوی ترکیبی از پارامترهای مدل است که تابع هزینه را برای کل مجموعه آموزشی به حداقل می‌رساند. این فرآیند، یک جستجو در «فضای پارامترها ($parameter\ space$)»ی مدل محسوب می‌شود. هرچه تعداد پارامترهای مدل بیشتر باشد، این فضا ابعاد بیشتری دارد و جستجو دشوارتر می‌شود؛ به‌عنوان مثال، پیدا کردن سوزن در یک انبار کاه ۳۰۰ بعدی بسیار سخت‌تر از پیدا کردن آن در یک فضای ۳ بعدی است. خوشبختانه، از آنجا که تابع هزینه در رگرسیون خطی محدب است، «سوزن» به‌سادگی در ته کاسه قرار دارد و یافتن آن آسان است.

© 2025 Mohsen Zare. All rights reserved.

گرادیان نزولی دسته‌ای(Batch Gradient Descent)

برای پیاده‌سازی گرادیان نزولی، باید گرادیان تابع هزینه را نسبت به هر پارامتر مدل θ_j محاسبه کنید. به عبارت دیگر، باید بررسی کنید که اگر تنها θ_j کمی تغییر کند، تابع هزینه چه مقدار تغییر خواهد کرد. این مقدار «مشتق جزئی (*partial derivative*)» نامیده می‌شود. می‌توان این کار را به این صورت تصور کرد: «اگر به سمت شرق حرکت کنم، شیب کوه زیر پایم چقدر است؟» سپس همین سوال را به سمت شمال و سایر ابعاد دیگر تکرار می‌کنیم (اگر بتوانید جهانی با بیش از سه بعد را تصور کنید). معادله ۴-۵ مشتق جزئی MSE را نسبت به پارامتر θ_j محاسبه می‌کند، که به صورت $(\partial \mathrm{MSE}(\boldsymbol{\theta}))/\partial \theta_j$ نشان داده می‌شود.

معادله ۴-۵. مشتقات جزئی تابع هزینه

$$\partial \mathrm{MSE}(\boldsymbol{\theta})/\partial \theta_j = 2\,m \sum_{i=1}^{m} \left(\theta^\top x^{(i)} - y^{(i)}\right) x_j^{(i)}$$

به جای محاسبه جداگانه این مشتقات جزئی، می‌توان از معادله ۴-۶ برای محاسبه همه آن‌ها به‌صورت هم‌زمان استفاده کرد. بردار گرادیان، که با $\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$ نمایش داده می‌شود، شامل تمام مشتقات جزئی تابع هزینه است، به‌طوری که برای هر پارامتر مدل یک مقدار وجود دارد.

معادله ۴-۶. بردار گرادیان تابع هزینه

$$\nabla_\theta \mathrm{MSE}(\theta) = \begin{bmatrix} \partial MSE(\theta)/\partial \theta_0 \\ \partial MSE(\theta)/\partial \theta_1 \\ \vdots \\ \partial MSE(\theta)/\partial \theta_\mathrm{n} \end{bmatrix} = \left(\frac{2}{m}\right) X^\mathrm{T} \left(X_\theta - y\right)$$

© 2025 Mohsen Zare. All rights reserved.

هشدار

توجه داشته باشید که این فرمول شامل محاسبات روی کل مجموعه آموزشی $\mathbf{X}$، در هر مرحله از گرادیان نزولی است! به همین دلیل، این الگوریتم «گرادیان نزولی دسته‌ای (*BGD: Batch Gradient Descent*)» نامیده می‌شود: در هر مرحله از تمام داده‌های آموزشی استفاده می‌کند (در واقع، نام دقیق‌تر ممکن است «گرادیان نزولی کامل (*full gradient descent*)» باشد). در نتیجه، این روش روی مجموعه‌های آموزشی بسیار بزرگ به‌طرز قابل‌توجهی کند است (به‌زودی با الگوریتم‌های گرادیان نزولی آشنا خواهید شد). با این حال، گرادیان نزولی نسبت به تعداد ویژگی‌ها مقیاس‌پذیر است؛ آموزش یک مدل رگرسیون خطی با صدها هزار ویژگی، با استفاده از گرادیان نزولی بسیار سریع‌تر از استفاده از معادله نرمال یا تجزیه SVD انجام می‌شود.

هنگامی که بردار گرادیان را که به سمت بالا اشاره دارد، به‌دست آوردید، کافی است در جهت مخالف حرکت کنید تا به سمت پایین تابع هزینه بروید. این به معنای کم کردن $\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$ از $\boldsymbol{\theta}$ است. در اینجا، نرخ یادگیری η وارد عمل می‌شود: بردار گرادیان در η ضرب می‌شود تا اندازه گام نزولی تعیین گردد (معادله ۴-۷).

معادله ۴-۷. گام گرادیان نزولی

$$\theta^{(t+1)}\ (next\ step) = \theta^{(t)} - \eta \nabla_{\theta} MSE\left(\theta^{(t)}\right)$$

بیایید نگاهی کوتاه به پیاده‌سازی این الگوریتم بیندازیم:

```
eta = 0.1 # learning rate
n_epochs = 1000
m = len(X_b) # number of instances

np.random.seed(42)
theta = np.random.randn(2, 1) # randomly initialized model parameters

for epoch in range(n_epochs):
  gradients = 2 / m * X_b.T @ (X_b @ theta - y)
```

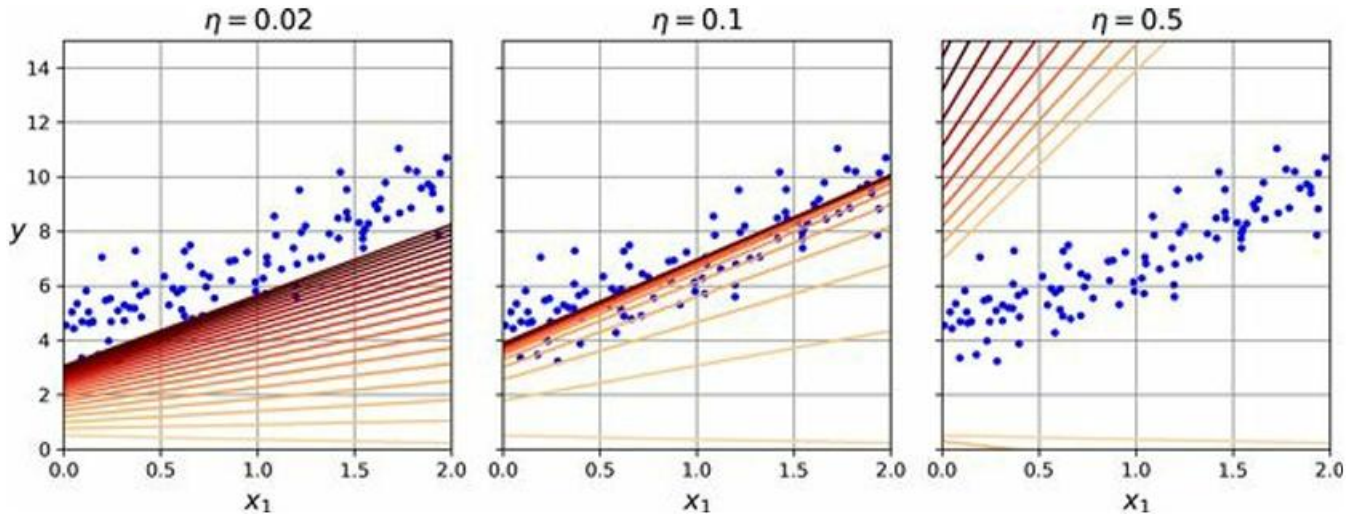
© 2025 Mohsen Zare. All rights reserved.

```
theta = theta - eta * gradients
```

کار دشواری نبود! هر بار که الگوریتم یک بار کامل روی کل مجموعه آموزشی اجرا می‌شود، یک «دوره ($epoch$)» نامیده می‌شود. حالا بیایید به مقادیر θ حاصل نگاهی بیندازیم:

```
>>> theta
array([[4.21509616],
[2.77011339]])
```

این دقیقاً همان نتیجه‌ای است که معادله نرمال به‌دست آورد! روش گرادیان نزولی به‌درستی عمل کرد. اما اگر از نرخ یادگیری (η : eta) متفاوتی استفاده می‌کردید، چه اتفاقی می‌افتاد؟ شکل ۴-۸، ۲۰ مرحله اول گرادیان نزولی را با سه نرخ یادگیری مختلف نشان می‌دهد. خط پایین هر نمودار، نقطه شروع تصادفی را مشخص می‌کند و هر دوره‌ی آموزشی (epoch) با یک خط تیره‌تر نمایش داده شده است.



شکل ۴-۸. گرادیان نزولی با نرخ‌های یادگیری مختلف

در سمت چپ، نرخ یادگیری بسیار پایین است: الگوریتم در نهایت به راه‌حل همگرا می‌شود، اما زمان زیادی طول می‌کشد. در وسط، نرخ یادگیری مناسب به نظر می‌رسد: تنها پس از چند دوره، الگوریتم به

© 2025 Mohsen Zare. All rights reserved.

راه‌حل همگرا می‌شود. در سمت راست، نرخ یادگیری بسیار بالا است: الگوریتم واگرا می‌شود، از مسیر خارج می‌شود و در هر مرحله از راه‌حل دورتر می‌گردد.

برای پیدا کردن نرخ یادگیری مناسب، می‌توان از «جستجوی شبکه‌ای (grid search)» استفاده کرد (به فصل ۲ مراجعه کنید). با این حال، ممکن است بخواهید تعداد دوره‌ها را محدود کنید تا مدل‌هایی که برای همگرایی بیش از حد زمان می‌برند، از جستجو حذف شوند.

شاید برایتان سؤال پیش بیاید که چگونه تعداد دوره‌ها را تنظیم کنیم. اگر تعداد دوره‌ها خیلی کم باشد، وقتی الگوریتم متوقف می‌شود، هنوز از راه‌حل بهینه فاصله زیادی خواهید داشت؛ اما اگر تعداد دوره‌ها خیلی زیاد باشد، در حالی که پارامترهای مدل دیگر تغییر نمی‌کنند، زمان هدر می‌رود. یک راه‌حل ساده این است که تعداد بسیار زیادی دوره تعیین کنید، اما الگوریتم را زمانی متوقف کنید که بردار گرادیان کوچک شود؛ یعنی وقتی نُرم آن از یک عدد کوچک ϵ (که به آن «خطای مجاز یا آستانه تحمل خطا ($tolerance$)» گفته می‌شود) کمتر شود. این وضعیت نشان می‌دهد که گرادیان نزولی تقریباً به حداقل خود رسیده است.

نرخ همگرایی

وقتی تابع هزینه محدب باشد و شیب آن به طور ناگهانی تغییر نکند (همان‌طور که در مورد تابع هزینه MSE صادق است)، الگوریتم گرادیان نزولیِ دسته‌ای با نرخ یادگیری ثابت در نهایت به راه‌حل بهینه همگرا می‌شود. با این حال، ممکن است نیاز باشد مدتی صبر کنید: بسته به شکل تابع هزینه، رسیدن به نقطه بهینه در محدوده ϵ می‌تواند به حدود $O(1/\epsilon)$ تکرار نیاز داشته باشد. اگر بخواهید راه‌حل دقیق‌تری داشته باشید و تلرانس را بر ۱۰ تقسیم کنید، الگوریتم ممکن است تقریباً ۱۰ برابر طولانی‌تر اجرا شود.

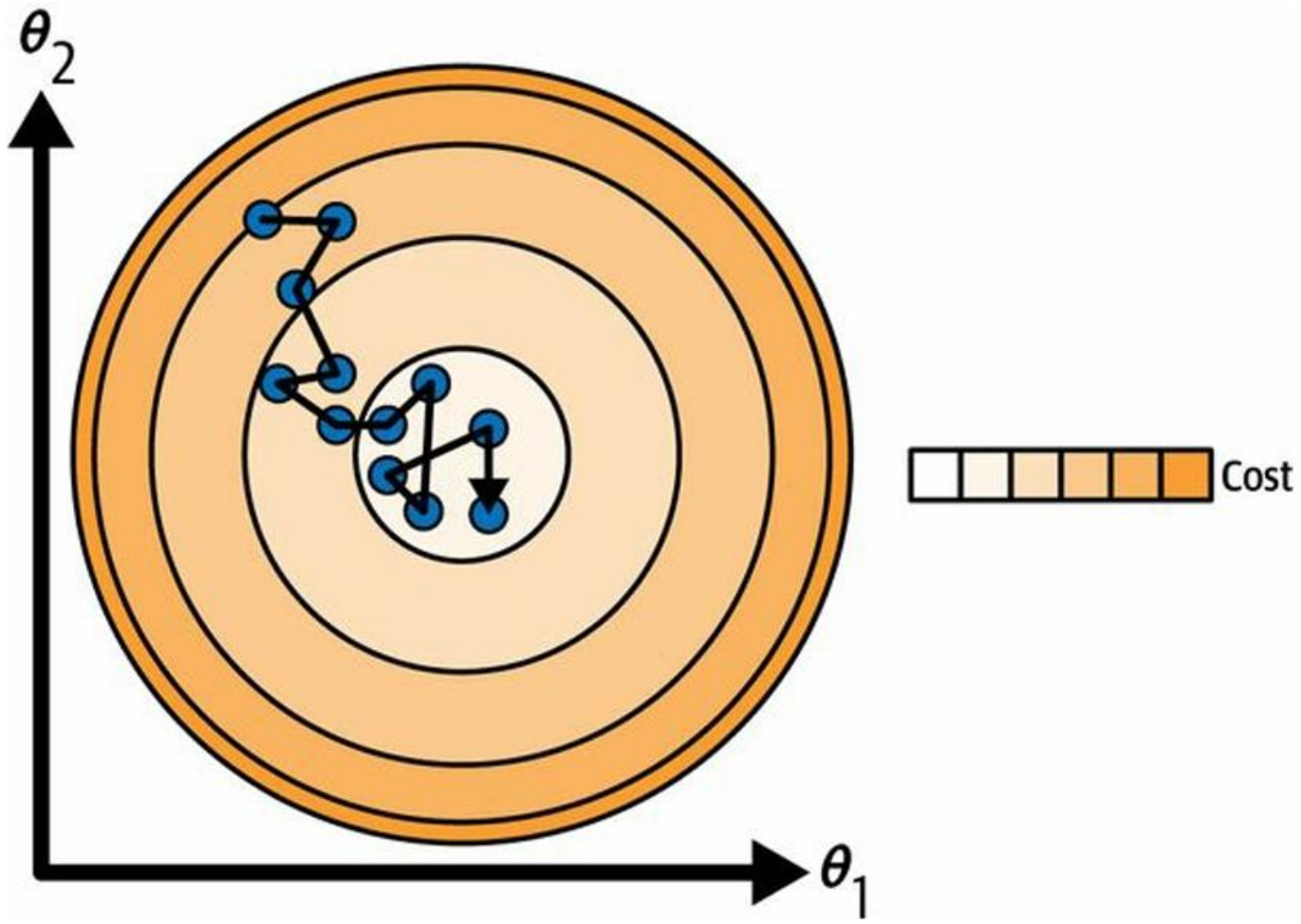
© 2025 Mohsen Zare. All rights reserved.

گرادیان نزولی تصادفی

مشکل اصلی گرادیان نزولی دسته‌ای این است که در هر مرحله از «کل مجموعه آموزشی» برای محاسبه گرادیان‌ها استفاده می‌کند، که باعث می‌شود الگوریتم روی مجموعه‌های آموزشی بزرگ بسیار کند شود. در مقابل، «گرادیان نزولی تصادفی (SGD: *stochastic gradient descent*)» در هر مرحله یک نمونه تصادفی از مجموعه آموزشی را انتخاب می‌کند و گرادیان را تنها بر اساس همان نمونه محاسبه می‌کند. کار کردن روی یک نمونه در هر زمان، الگوریتم را بسیار سریع‌تر می‌کند، زیرا در هر تکرار داده‌های کمی برای پردازش وجود دارد. علاوه بر این، SGD امکان آموزش روی مجموعه‌های آموزشی عظیم را فراهم می‌کند، زیرا در هر تکرار تنها یک نمونه باید در حافظه قرار گیرد (گرادیان نزولی تصادفی می‌تواند به‌صورت الگوریتم خارج از هسته پیاده‌سازی شود؛ به فصل ۱ مراجعه کنید).

از سوی دیگر، به دلیل ماهیت تصادفی (یعنی رندوم بودن)، این الگوریتم نسبت به گرادیان نزولی دسته‌ای بسیار نامنظم‌تر عمل می‌کند: تابع هزینه به جای کاهش تدریجی تا رسیدن به مینیمم، بالا و پایین می‌رود و تنها به طور متوسط کاهش می‌یابد. با گذشت زمان، مقدار تابع هزینه به مینیمم نزدیک می‌شود، اما حتی پس از رسیدن، نوسان ادامه می‌یابد و هرگز به حالت آرام کامل بازنمی‌گردد (شکل ۹-۴ را ببینید). وقتی الگوریتم متوقف شود، مقادیر نهایی پارامترها مناسب هستند، اما دقیقاً بهینه نیستند.

© 2025 Mohsen Zare. All rights reserved.



شکل ۴-۹. با روش گرادیان نزولی تصادفی، هر مرحله آموزش بسیار سریع‌تر است، اما در عین حال بسیار تصادفی‌تر از زمانی است که از روش گرادیان نزولی دسته‌ای استفاده می‌شود.

وقتی تابع هزینه بسیار نامنظم باشد (مانند شکل ۴-۶)، این ویژگی می‌تواند در واقع به الگوریتم کمک کند تا از مینیمم‌های محلی خارج شود. بنابراین، گرادیان نزولی تصادفی شانس بیشتری برای رسیدن به مینیمم مطلق نسبت به گرادیان نزولی دسته‌ای دارد. به عبارت دیگر، ماهیت تصادفی الگوریتم برای فرار از بهینه محلی مفید است، اما معایبی هم دارد؛ مهم‌ترین آن این است که الگوریتم هرگز نمی‌تواند دقیقاً در حداقل قرار گیرد. یک راه حل برای این مشکل، کاهش تدریجی نرخ یادگیری است: مراحل ابتدا بزرگ هستند (که به پیشرفت سریع و فرار از مینیمم‌های محلی کمک می‌کند) و سپس به تدریج کوچک‌تر می‌شوند تا الگوریتم بتواند در مینیمم مطلق مستقر شود. این فرآیند مشابه «شبیه‌سازی تبرید (*simulated annealing*)» است— الگوریتم فرا ابتکاری برای جستجوی سراسری؛ برای فرار از مینیمم‌های محلی کاربرد دارد—، الگوریتمی الهام‌گرفته از فرآیند سرد شدن فلزات(Annealing) در علم مواد(متالورژی)، که در آن فلز مذاب به آرامی سرد می‌شود. تابعی که نرخ یادگیری را در هر تکرار مشخص

© 2025 Mohsen Zare. All rights reserved.

می‌کند، «زمان‌بندیِ نرخ یادگیری (*learning schedule*)» نامیده می‌شود. اگر نرخ یادگیری خیلی سریع کاهش یابد، ممکن است الگوریتم در یک مینیمم محلی گیر کند یا حتی قبل از رسیدن به مینیمم مطلق، منجمد شود.

اگر نرخ یادگیری خیلی آهسته کاهش یابد، الگوریتم ممکن است برای مدت طولانی حول مینیمم نوسان کند و در صورت توقف زودهنگام آموزش، در نهایت به «راه‌حل غیربهینه» برسد.

کد زیر، گرادیان نزولی تصادفی را با استفاده از یک برنامه زمانی یادگیری ساده پیاده‌سازی می‌کند:

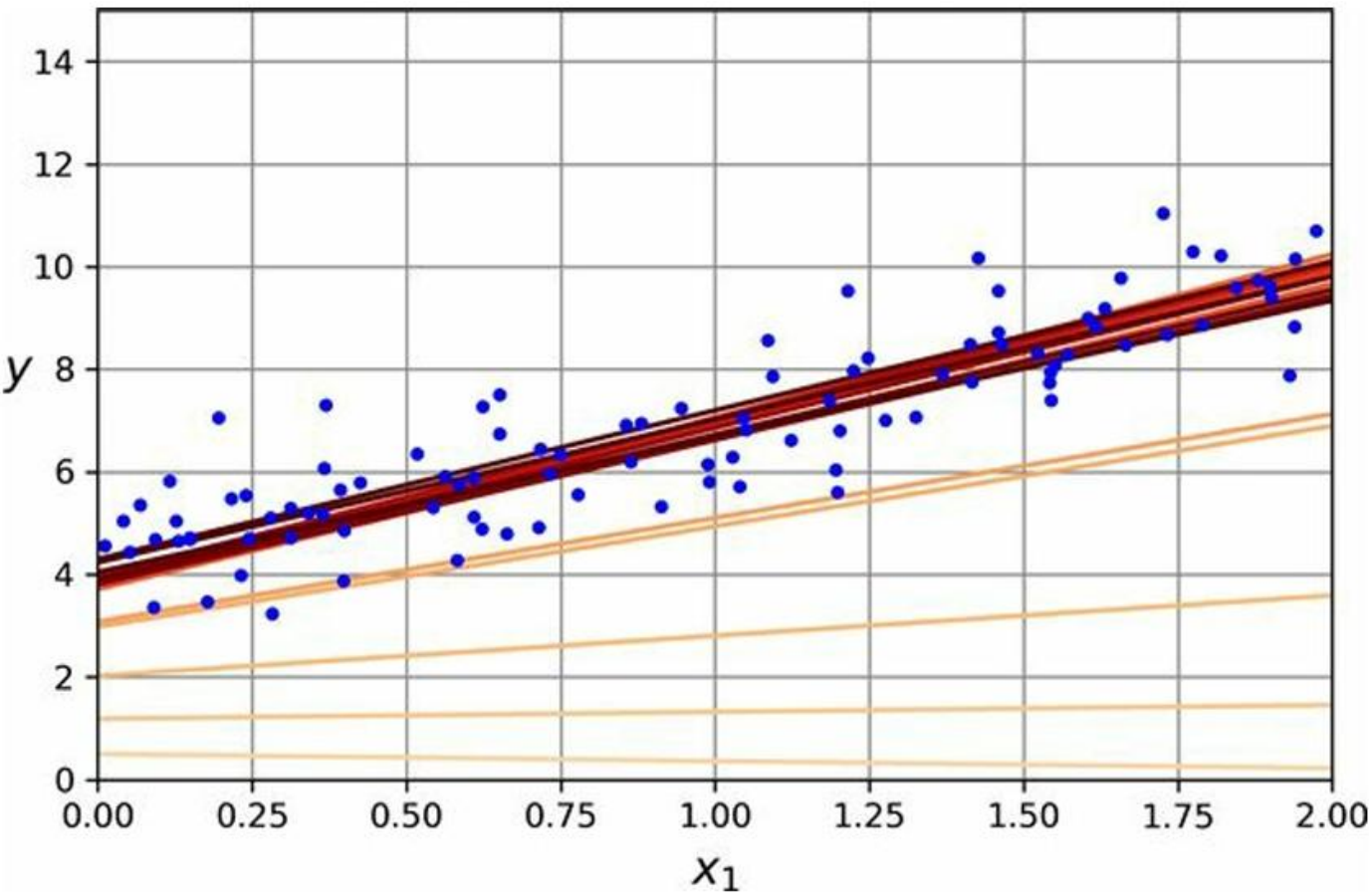
```
n_epochs = 50
t0, t1 = 5, 50 # learning schedule hyperparameters

def learning_schedule(t):
  return t0 / (t + t1)

np.random.seed(42)
theta = np.random.randn(2, 1) # random initialization

for epoch in range(n_epochs):
  for iteration in range(m):
    random_index = np.random.randint(m)
    xi = X_b[random_index : random_index + 1]
    yi = y[random_index : random_index + 1]
    gradients = 2 * xi.T @ (xi @ theta - yi) # for SGD, do not divide by m
    eta = learning_schedule(epoch * m + iteration)
    theta = theta - eta * gradients
```

طبق قرارداد، این فرآیند را در چندین دور m تکرار می‌کنیم؛ هر دور(round)، مانند قبل، یک «دوره (*epoch*)» نامیده می‌شود. در حالی که کد گرادیان نزولی دسته‌ای ۱,۰۰۰ بار روی کل مجموعه آموزشی اجرا می‌شد، این کد تنها ۵۰ بار روی مجموعه آموزشی تکرار می‌شود و به یک راه‌حل نسبتاً خوب می‌رسد:

```
>>> theta
array([[4.21076011],
[2.74856079]])
```

© 2025 Mohsen Zare. All rights reserved.

شکل ۴-۱۰، بیست مرحله اول آموزش را نشان می‌دهد (به نامنظم بودن مراحل توجه کنید). توجه داشته باشید که از آنجا که نمونه‌ها به صورت تصادفی انتخاب می‌شوند، برخی از نمونه‌ها ممکن است چندین بار در هر دوره استفاده شوند، در حالی که برخی دیگر ممکن است اصلاً انتخاب نشوند. اگر بخواهید اطمینان حاصل کنید که الگوریتم در هر دوره از تمام نمونه‌ها عبور کند، یک رویکرد دیگر این است که مجموعه آموزشی را درهم و مخلوط(shuffle) کنید (توجه داشته باشید که ویژگی‌ها و برچسب‌ها باید به‌طور مشترک درهم زده شوند)، سپس نمونه‌ها را یک به یک پردازش کنید و پس از پایان دوره دوباره مجموعه را درهم‌ریخته و ادامه دهید. با این حال، این روش پیچیده‌تر است و معمولاً نتیجه قابل‌توجهی بهتر ارائه نمی‌دهد.



شکل ۴-۱۰. ۲۰ مرحله ی اولِ گرادیان نزولی تصادفی

© 2025 Mohsen Zare. All rights reserved.

هشدار

هنگام استفاده از گرادیان نزولی تصادفی (SGD)، نمونه‌های آموزشی باید مستقل و با توزیع یکسان (IID) باشند تا اطمینان حاصل شود که پارامترها به طور متوسط به سمت بهینه سراسری کشیده می‌شوند. یک روش ساده برای تضمین این امر، مخلوط کردن نمونه‌ها در طول آموزش است؛ برای مثال، می‌توان هر نمونه را به صورت تصادفی انتخاب کرد یا مجموعه آموزشی را در ابتدای هر دوره shuffle کرد. اگر نمونه‌ها را جابه‌جا نکنید – مثلاً اگر نمونه‌ها بر اساس برچسب مرتب شده باشند – الگوریتم SGD ابتدا روی یک برچسب بهینه‌سازی می‌کند، سپس برچسب بعدی و غیره، و در نهایت ممکن است نزدیک مینیمم سراسری قرار نگیرد.

برای انجام رگرسیون خطی با استفاده از گرادیان نزولی تصادفی (SGD) در Scikit-Learn، می‌توانید از کلاس SGDRegressor استفاده کنید که به طور پیش‌فرض تابع هزینه MSE را بهینه‌سازی می‌کند. کد زیر برای حداکثر ۱,۰۰۰ دوره (max_iter) یا تا زمانی که کاهش زیان در طول ۱۰۰ دوره متوالی (n_iter_no_change) کمتر از $۱۰^{-۵}$ (tol) باشد، اجرا می‌شود. الگوریتم با نرخ یادگیری ۰٫۰۱ (eta0) و با استفاده از برنامه یادگیری پیش‌فرض (که متفاوت از برنامه‌ای است که ما استفاده کردیم) شروع می‌شود. در نهایت، هیچ گونه منظم‌سازی (penalty=None) اعمال نمی‌شود؛ جزئیات بیشتر در مورد منظم‌سازی به زودی ارائه خواهد شد:

```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(max_iter=1000, tol=1e-5, penalty=None, eta0=0.01,
                       n_iter_no_change=100, random_state=42)
sgd_reg.fit(X, y.ravel()) # y.ravel() because fit() expects 1D targets
```

یک بار دیگر، مشاهده می‌کنید که الگوریتم راه‌حلی بسیار نزدیک به نتیجه‌ای که معادله نرمال ارائه می‌دهد، پیدا کرده است:

```
>>> sgd_reg.intercept_, sgd_reg.coef_
```

© 2025 Mohsen Zare. All rights reserved.

```
(array([4.21278812]), array([2.77270267]))
```

نکته

تمام تخمین‌گرهای Scikit-Learn را می‌توان با استفاده از متد fit() آموزش داد. برخی از تخمین‌گرها همچنین متد partial_fit() را ارائه می‌دهند که امکان اجرای یک دور آموزش روی یک یا چند نمونه را فراهم می‌کند. این متد هایپرپارامترهایی مانند max_iter یا tol را نادیده می‌گیرد و فراخوانی مکرر آن به تدریج مدل را آموزش می‌دهد. این روش زمانی مفید است که نیاز به کنترل دقیق‌تر بر فرآیند آموزش دارید. برخی مدل‌ها به جای آن، یا علاوه بر آن، دارای هایپرپارامتر warm_start هستند: اگر warm_start=True تنظیم شود، فراخوانی متد fit() روی یک مدل آموزش‌دیده مدل را بازنشانی نمی‌کند و آموزش را از جایی که متوقف شده بود ادامه می‌دهد، با رعایت هایپرپارامترهایی مانند max_iter و tol. لازم به ذکر است که fit() شمارنده تکرار برنامه یادگیری را بازنشانی می‌کند، در حالی که partial_fit() این کار را انجام نمی‌دهد.

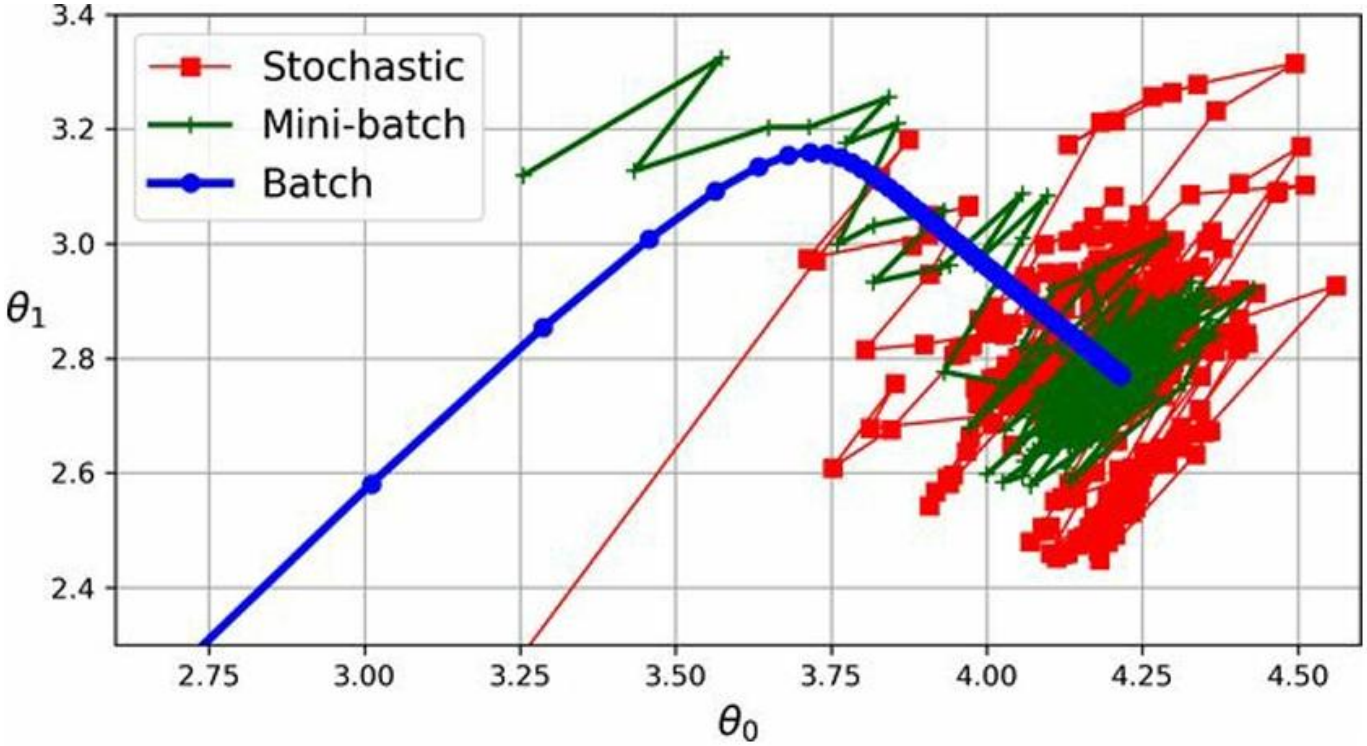
© 2025 Mohsen Zare. All rights reserved.

الگوریتم گرادیان نزولی مینی دسته ای

آخرین الگوریتم گرادیان نزولی که بررسی می‌کنیم، «گرادیان نزولی مینی دسته‌ای (*gradient descent mini-batch*)» است. وقتی با گرادیان نزولی دسته‌ای و گرادیان نزولی تصادفی آشنا باشید، درک آن ساده است: در هر مرحله، به جای محاسبه گرادیان‌ها بر اساس کل مجموعه آموزشی (مانند GD دسته‌ای) یا تنها یک نمونه (مانند GD تصادفی)، mini-batch GD گرادیان‌ها را روی مجموعه‌های تصادفی کوچکی از نمونه‌ها، به نام «مینی دسته ها (*mini-batches*)» ، محاسبه می‌کند. مزیت اصلی الگوریتم‌های mini-batch نسبت به stochastic GD این است که با بهره‌گیری از بهینه‌سازی ماتریسی و سخت‌افزار مناسب، به ویژه هنگام استفاده از پردازنده‌های گرافیکی (GPU)، می‌توان عملکرد آموزش را به طور قابل توجهی افزایش داد.

پیشرفت الگوریتم در فضای پارامتر، به ویژه وقتی از مینی دسته های نسبتاً بزرگ استفاده شود، نسبت به GD تصادفی کمتر نوسانی است. در نتیجه، GD مینی دسته ای در نهایت کمی نزدیک‌تر به مینیمم حرکت می‌کند، اما ممکن است فرار از مینیمم‌های محلی برای آن دشوارتر باشد (این مسئله در مسائلی که چندین مینیمم محلی دارند اهمیت دارد، برخلاف رگرسیون خطی با تابع هزینه MSE). شکل ۴-۱۱ مسیرهای طی شده توسط سه الگوریتم گرادیان نزولی را در فضای پارامتر نشان می‌دهد. همه آن‌ها به نزدیکی مینیمم می‌رسند، اما مسیر گرادیان نزولی دسته‌ای دقیقاً در مینیمم متوقف می‌شود، در حالی که گرادیان نزولی تصادفی و گرادیان نزولی مینی دسته ای به حرکت خود ادامه می‌دهند. با این حال، توجه داشته باشید که گرادیان نزولی دسته‌ای برای طی کردن هر مرحله زمان بیشتری نیاز دارد و اگر از یک برنامه یادگیری مناسب استفاده شود، هر دو الگوریتم تصادفی و مینی دسته‌ای نیز به مینیمم همگرا خواهند شد.

© 2025 Mohsen Zare. All rights reserved.



شکل ۴-۱۱. مسیرهای گرادیان نزولی در فضای پارامتر

جدول ۴-۱ الگوریتم‌هایی را که تاکنون برای رگرسیون خطی بررسی کرده‌ایم، مقایسه می‌کند. در این جدول، m تعداد نمونه‌های آموزشی و n تعداد ویژگی‌ها در نظر گرفته شده است.

© 2025 Mohsen Zare. All rights reserved.

جدول ۴-۱. مقایسه الگوریتم‌های رگرسیون خطی

الگوریتم	m بزرگ	پشتیبانی خارج از هسته	n بزرگ	هایپرپارامترها
معادله نرمال	سریع	خیر	آهسته	۰
SVD	سریع	خیر	آهسته	۰
Batch GD	آهسته	خیر	سریع	۲
Stochastic GD	سریع	بله	سریع	≥۲
Mini-batch GD	سریع	بله	سریع	≥۲

تقریباً پس از اتمام آموزش، تفاوت قابل توجهی میان این الگوریتم‌ها وجود ندارد: همه آن‌ها در نهایت به مدل‌های بسیار مشابه می‌رسند و پیش‌بینی‌ها را به همان شیوه انجام می‌دهند.

© 2025 Mohsen Zare. All rights reserved.

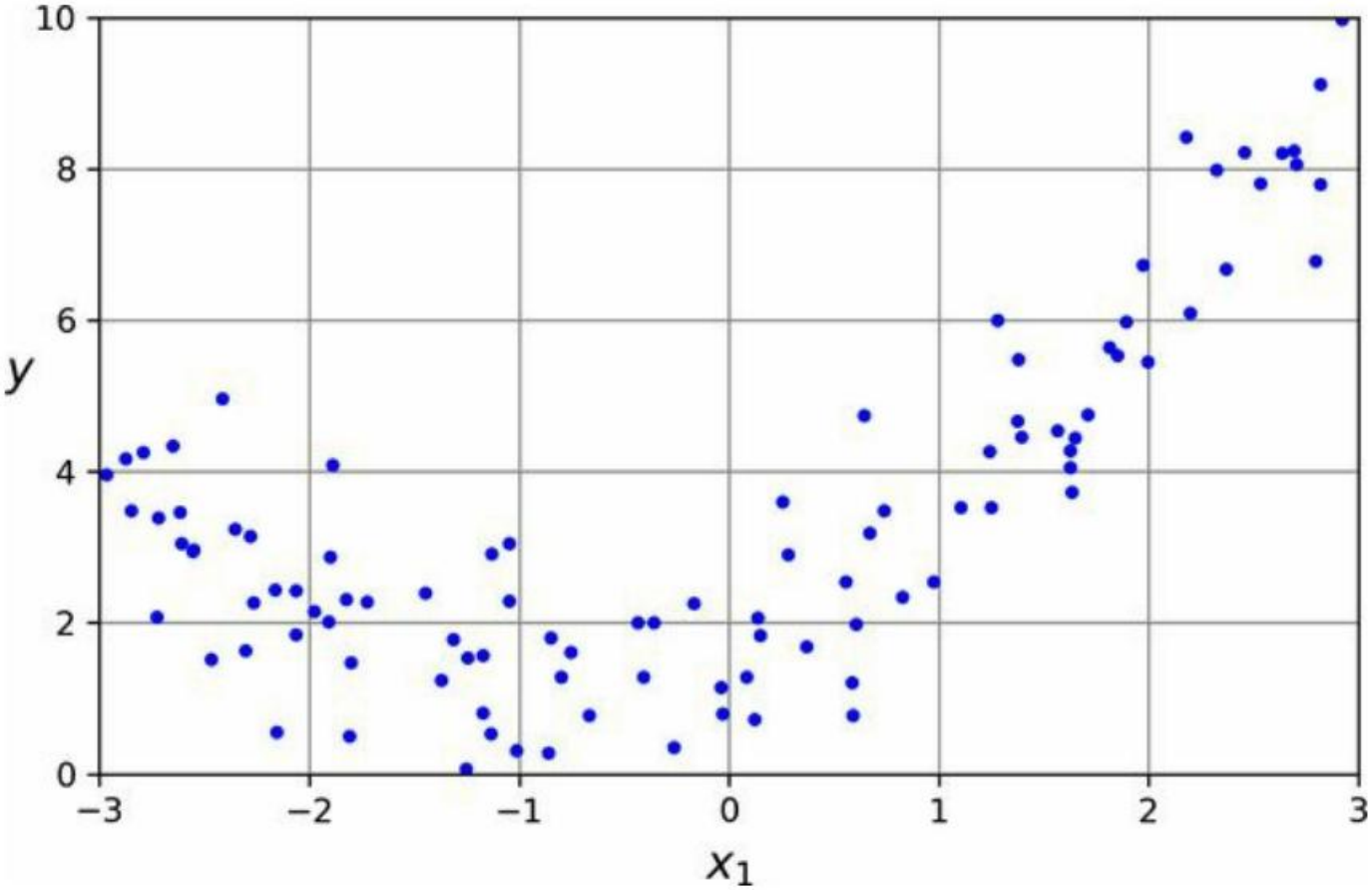
رگرسیون چندجمله‌ای

اگر داده‌های شما پیچیده‌تر از یک خط مستقیم باشند، چه کاری می‌توان انجام داد؟ جالب است که می‌توان از یک مدل خطی برای برازش داده‌های غیرخطی استفاده کرد. یک روش ساده برای این کار، ایجاد ویژگی‌های جدید از طریق توان‌های هر ویژگی موجود و سپس آموزش یک مدل خطی روی این مجموعه گسترده از ویژگی‌ها است. این تکنیک به نام «رگرسیون چندجمله‌ای (*polynomial regression*)» شناخته می‌شود.

بیایید یک مثال عملی را بررسی کنیم. ابتدا، با استفاده از یک «معادله درجه دوم ساده (*quadratic equation*)»—که شکلی مانند $y = ax^2 + bx + c$ دارد—و افزودن مقداری نویز، مجموعه‌ای از داده‌های غیرخطی تولید می‌کنیم (شکل ۴-۱۲ را مشاهده کنید).

```
np.random.seed(42)
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
```

© 2025 Mohsen Zare. All rights reserved.



شکل ۴-۱۲. مجموعه داده غیرخطی و نویزی تولید شده

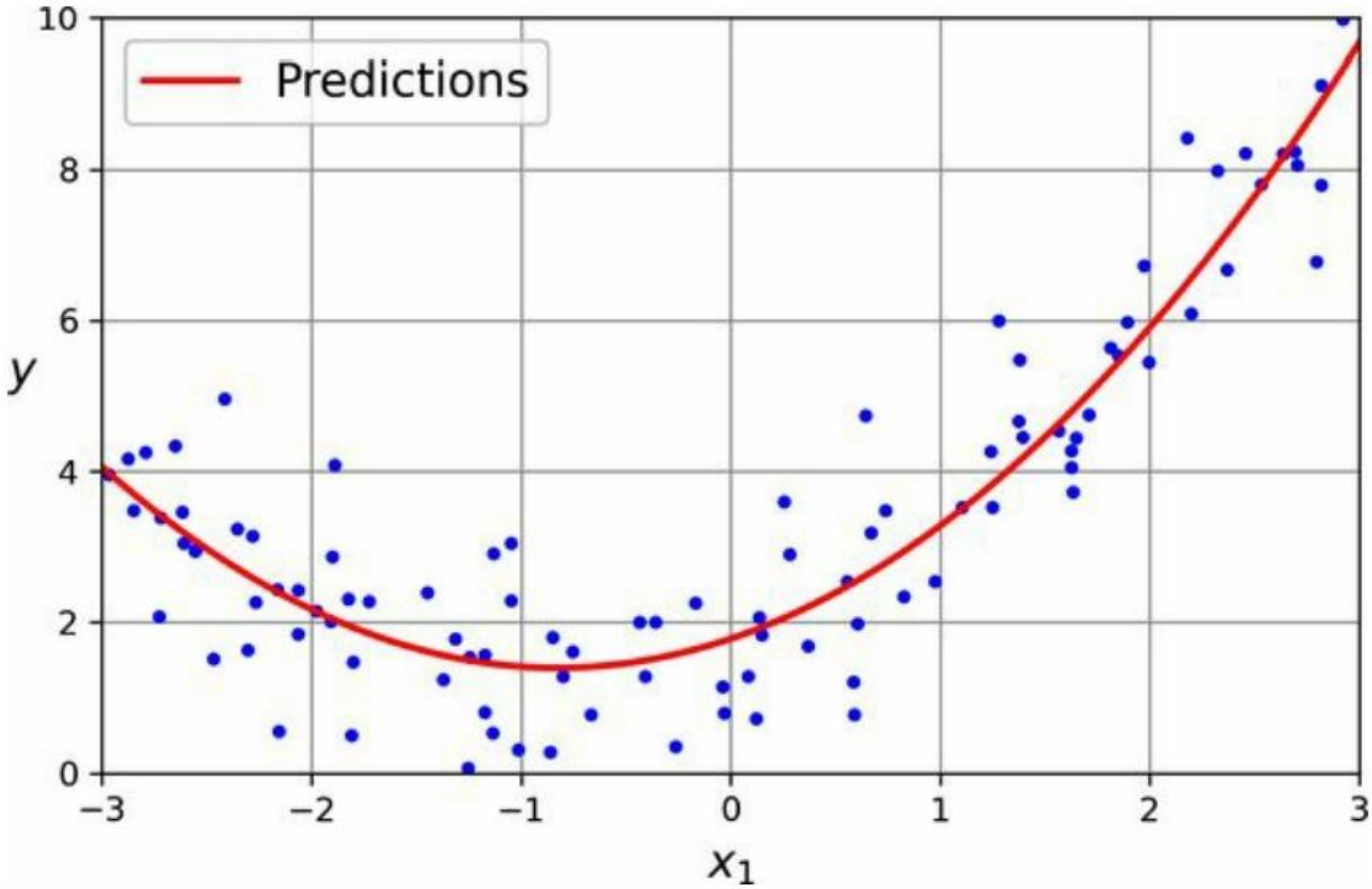
واضح است که یک خط مستقیم نمی‌تواند این داده‌ها را به درستی مدل کند. بنابراین، از کلاس PolynomialFeatures در Scikit-Learn استفاده می‌کنیم تا ویژگی‌های آموزشی را تبدیل کنیم و توان دوم هر ویژگی را به عنوان یک ویژگی جدید به مجموعه اضافه کنیم (در این مثال، تنها یک ویژگی وجود دارد):

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly_features = PolynomialFeatures(degree=2, include_bias=False)
>>> X_poly = poly_features.fit_transform(X)
>>> X[0]
array([-0.75275929])
>>> X_poly[0]
array([-0.75275929, 0.56664654])
```

© 2025 Mohsen Zare. All rights reserved.

اکنون X_poly شامل ویژگی اصلی X به علاوه توان دوم آن است. می‌توانیم یک مدل LinearRegression را روی این داده‌های توسعه‌یافته برازش دهیم (شکل ۴-۱۳).

```
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```



شکل ۴-۱۳. پیش‌بینی‌های مدل رگرسیون چندجمله‌ای

© 2025 Mohsen Zare. All rights reserved.

نتایج خوب هستند: مدل، تقریباً تابع $\hat{y} = 0.56\, x^{12} + 0.93\, x^{1} + 1.78$ را تخمین می‌زند در حالی که تابع اصلی داده‌ها به صورت: $y = 0.5\, x^{12} + 0.93\, x^{1} + 1.78 +$ Gaussian noise بود.

توجه داشته باشید که وقتی چندین ویژگی وجود دارد، رگرسیون چندجمله‌ای می‌تواند روابط میان ویژگی‌ها را شناسایی کند—کاری که یک مدل رگرسیون خطی ساده قادر به انجام آن نیست. این قابلیت از آنجا ناشی می‌شود که PolynomialFeatures تمام ترکیب‌های ویژگی‌ها را تا درجه مشخص شده (degree) ایجاد می‌کند. به عنوان مثال، اگر دو ویژگی a و b وجود داشته باشد و درجه ۳ در نظر گرفته شود، PolynomialFeatures نه تنها ویژگی‌های a^2، a^3، b^2 و b^3 را اضافه می‌کند، بلکه ترکیباتی مانند ab، a^2b و ab^2 را نیز شامل می‌شود.

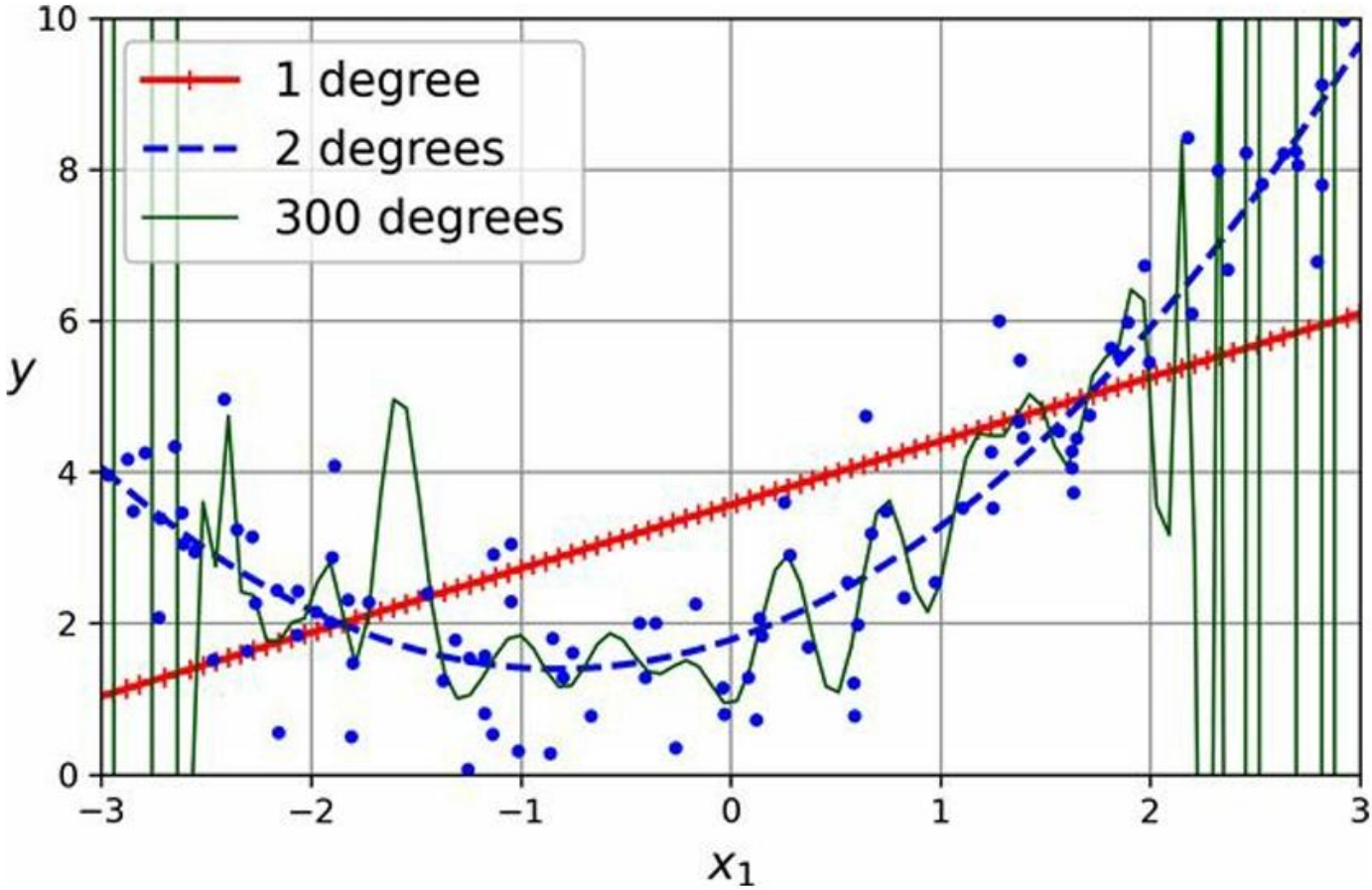
هشدار

PolynomialFeatures(degree=d) یک آرایه حاوی n ویژگی را به آرایه‌ای با $(n+d)!\,/\,d!\,n!$ ویژگی تبدیل می‌کند، که در آن $n!$ فاکتوریل n است که برابر با $1 \times 2 \times 3 \times \ldots \times n$ می‌باشد. به این ترتیب، مراقب افزایش ترکیبی تعداد ویژگی‌ها باشید!

© 2025 Mohsen Zare. All rights reserved.

منحنی‌های یادگیری

اگر یک رگرسیون چندجمله‌ای با درجه بالا انجام دهید، معمولاً داده‌های آموزشی را به مراتب بهتر از رگرسیون خطی ساده برازش می‌دهد. به عنوان مثال، شکل ۴-۱۴ یک مدل چندجمله‌ای درجه ۳۰۰ را روی داده‌های آموزشی قبلی اعمال کرده و آن را با یک مدل خطی ساده و یک مدل درجه دوم (چندجمله‌ای درجه ۲) مقایسه می‌کند. مشاهده کنید که مدل چندجمله‌ای ۳۰۰ درجه چگونه برای نزدیک شدن هرچه بیشتر به نمونه‌های آموزشی، شدیداً تغییر می‌کند.



شکل ۴-۱۴. رگرسیون چندجمله‌ای درجه بالا

این مدل چندجمله‌ای با درجه بالا به شدت داده‌های آموزشی را بیش‌برازش می‌کند، در حالی که مدل خطی ساده داده‌ها را کم‌برازش می‌کند. مدل درجه دوم بهترین تعمیم را ارائه می‌دهد که منطقی است، زیرا داده‌ها با یک مدل درجه دوم تولید شده‌اند.

© 2025 Mohsen Zare. All rights reserved.

با این حال، به طور کلی ما تابع واقعی تولید داده‌ها را نمی‌دانیم. بنابراین، چگونه می‌توان تعیین کرد که مدل چقدر پیچیده باشد؟ و چگونه می‌توان تشخیص داد که مدل داده‌ها را بیش‌برازش کرده یا کمتر از حد برازش کرده است؟

در فصل ۲، از اعتبارسنجی متقاطع برای برآورد عملکرد تعمیم (generalization performance) مدل استفاده کردید. اگر مدل روی داده‌های آموزشی عملکرد خوبی داشته باشد اما در اعتبارسنجی متقاطع عملکرد ضعیفی نشان دهد، مدل بیش‌برازش کرده است. اگر مدل روی هر دو مجموعه، هم داده‌های آموزشی و هم اعتبارسنجی، عملکرد ضعیفی داشته باشد، نشان‌دهنده کم‌برازش است. این روش، یکی از راه‌های تشخیص میزان پیچیدگی مناسب مدل و جلوگیری از بیش‌برازش یا کم‌برازش محسوب می‌شود.

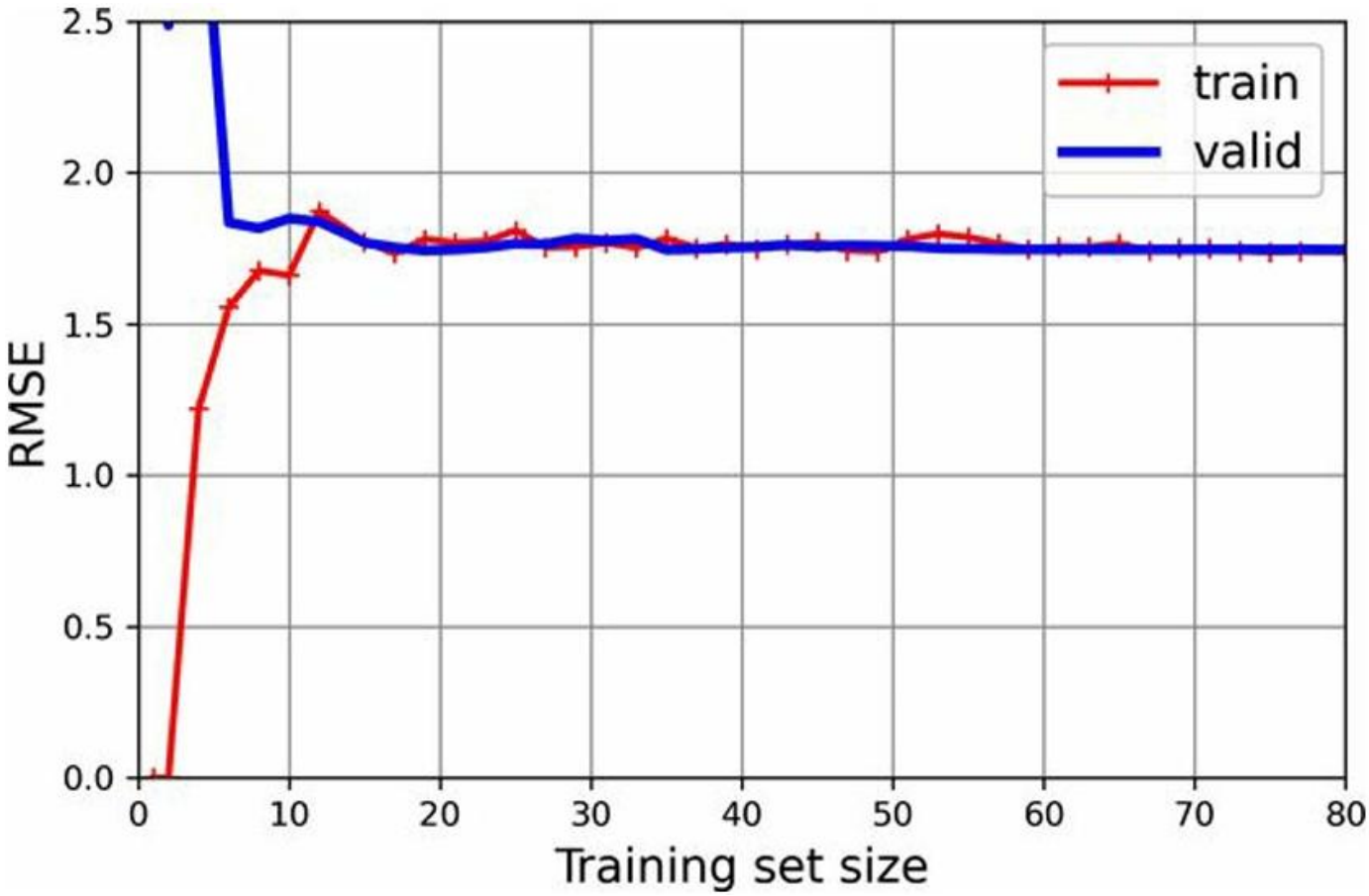
راه دیگر برای تشخیص بیش‌برازش یا کم‌برازش، بررسی «منحنی‌های یادگیری (*learning curves*)» است. این منحنی‌ها، «خطای آموزش» و «خطای اعتبارسنجی» مدل را به‌عنوان تابعی از پیشرفت آموزش نشان می‌دهند. برای این کار، مدل را در فواصل منظم در طول آموزش، هم روی مجموعه آموزش و هم روی مجموعه اعتبارسنجی ارزیابی کرده و نتایج را رسم می‌کنیم. اگر مدل امکان آموزش تدریجی نداشته باشد (یعنی از متدهایی مانند partial_fit یا ویژگی warm_start پشتیبانی نکند)، باید مدل را چندین بار روی زیرمجموعه‌های بزرگ‌تر مجموعه آموزشی آموزش دهید تا منحنی‌های یادگیری را به دست آورید.

کتابخانه Scikit-Learn یک تابع مفید به نام learning_curve() ارائه می‌دهد که می‌تواند در ترسیم منحنی‌های یادگیری کمک کند. این تابع مدل را با استفاده از اعتبارسنجی متقاطع آموزش داده و ارزیابی می‌کند. به‌طور پیش‌فرض، مدل با استفاده از زیرمجموعه‌های افزایشی مجموعه آموزشی دوباره آموزش داده می‌شود. اما اگر مدل از یادگیری افزایشی (incremental learning) پشتیبانی کند، می‌توانید با تنظیم پارامتر exploit_incremental_learning=True هنگام فراخوانی تابع، مدل را به‌صورت افزایشی آموزش دهید. این تابع اندازه‌های مجموعه آموزشی مورد استفاده، امتیازها آموزش و امتیازها اعتبارسنجی را برای هر اندازه و هر زیربخش (fold) اعتبارسنجی متقاطع بازمی‌گرداند. می‌توانیم از این تابع برای بررسی منحنی‌های یادگیری مدل رگرسیون خطی ساده استفاده کنیم (شکل ۱۵-۴).

```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, valid_scores = learning_curve(
```

© 2025 Mohsen Zare. All rights reserved.

```
  LinearRegression(), X, y, train_sizes=np.linspace(0.01, 1.0, 40), cv=5,
  scoring="neg_root_mean_squared_error")
train_errors = -train_scores.mean(axis=1)
valid_errors = -valid_scores.mean(axis=1)

plt.plot(train_sizes, train_errors, "r-+", linewidth=2, label="train")
plt.plot(train_sizes, valid_errors, "b-", linewidth=3, label="valid")
[...] # beautify the figure: add labels, axis, grid, and legend
plt.show()
```



شکل ۴-۱۵. منحنی‌های یادگیری

این مدل کم‌برازش دارد. برای درک دلیل آن، ابتدا به خطای آموزش نگاه می‌کنیم. وقتی تنها یک یا دو نمونه در مجموعه آموزشی وجود دارد، مدل می‌تواند آن‌ها را تقریباً به‌طور کامل برازش دهد، بنابراین منحنی خطای آموزش از صفر شروع می‌شود. با افزایش تعداد نمونه‌ها، برازش کامل داده‌های آموزشی برای مدل غیرممکن می‌شود؛ هم به دلیل وجود نویز در داده‌ها و هم به دلیل این که رابطه واقعی

© 2025 Mohsen Zare. All rights reserved.

داده‌ها خطی نیست. در نتیجه، خطای آموزش افزایش یافته و در نهایت به یک سطح ثابت می‌رسد، جایی که افزودن نمونه‌های جدید تغییر قابل توجهی در میانگین خطای آموزش ایجاد نمی‌کند. حال به خطای اعتبارسنجی می‌پردازیم. وقتی مدل روی تعداد کمی نمونه آموزش می‌بیند، قادر به تعمیم مناسب نیست و به همین دلیل خطای اعتبارسنجی در ابتدا بسیار زیاد است. با افزایش تعداد نمونه‌های آموزشی، مدل به تدریج یاد می‌گیرد و خطای اعتبارسنجی کاهش می‌یابد. با این حال، از آنجا که یک خط مستقیم نمی‌تواند ساختار داده‌ها را به خوبی مدل کند، خطای اعتبارسنجی نیز در نهایت به سطحی تقریباً ثابت نزدیک به منحنی خطای آموزش می‌رسد.

این منحنی‌های یادگیری نمونه‌ای واضح از کم‌برازش هستند. هر دو منحنی—خطای آموزش و خطای اعتبارسنجی—به یک سطح ثابت رسیده‌اند و این سطح، نسبتاً بالا و دور از خطای بهینه است. به عبارت دیگر، مدل قادر به یادگیری کافی از داده‌ها نبوده و تعمیم ضعیفی دارد.

نکته

وقتی مدل کم‌برازش دارد، اضافه کردن داده‌های آموزشی بیشتر معمولاً تغییری ایجاد نمی‌کند، چون مشکل از ظرفیت مدل یا ویژگی‌های ارائه‌شده است، نه کمبود داده. برای رفع این مشکل، باید:

۱. مدل پیچیده‌تر انتخاب کنید (مثلاً رگرسیون چندجمله‌ای به جای خطی ساده یا شبکه عصبی با لایه‌های بیشتر).

۲. ویژگی‌های بهتر یا جدید به مدل بدهید تا اطلاعات بیشتری از داده‌ها استخراج شود.

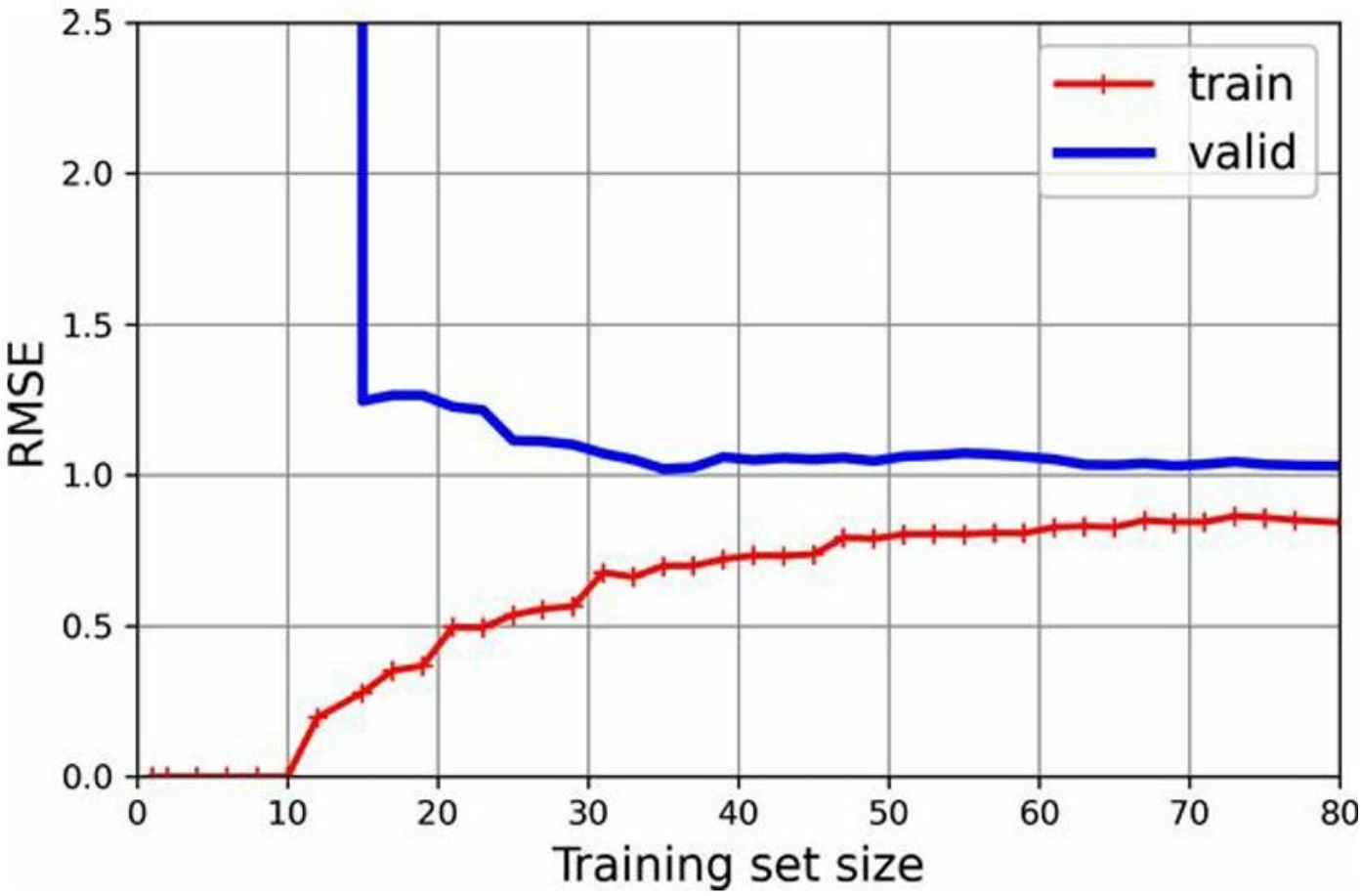
به عبارت دیگر، بدون تغییر ساختار مدل یا بهبود ویژگی‌ها، خطای آموزش و اعتبارسنجی همچنان بالا باقی می‌ماند.

اکنون به منحنی‌های یادگیری یک مدل چندجمله‌ای درجه دهم روی همان داده‌ها نگاه می‌کنیم (شکل ۱۶-۴):

```
from sklearn.pipeline import make_pipeline
```

© 2025 Mohsen Zare. All rights reserved.

```
polynomial_regression = make_pipeline(
  PolynomialFeatures(degree=10, include_bias=False),
  LinearRegression())

train_sizes, train_scores, valid_scores = learning_curve(
  polynomial_regression, X, y, train_sizes=np.linspace(0.01, 1.0, 40), cv=5,
  scoring="neg_root_mean_squared_error")
[...] # same as earlier
```



شکل ۴-۱۶. منحنی‌های یادگیری برای مدل چندجمله‌ای درجه دهم

این منحنی‌های یادگیری تا حدی شبیه به نمونه‌های قبلی هستند، اما دو تفاوت مهم دارند:

© 2025 Mohsen Zare. All rights reserved.

- کاهش چشمگیر خطای داده‌های آموزشی: مدل اکنون داده‌های آموزشی را بسیار بهتر برازش می‌دهد.
- فاصله بین منحنی‌ها: خطای اعتبارسنجی به‌طور قابل توجهی بالاتر از خطای آموزش است، که نشانه‌ی بیش‌برازش مدل است. با این حال، اگر مجموعه آموزشی بسیار بزرگ‌تر شود، این دو منحنی به تدریج به یکدیگر نزدیک‌تر خواهند شد.

نکته

یکی از راه‌های کاهش بیش‌برازش، افزایش تعداد نمونه‌های آموزشی است؛ با افزودن داده‌های بیشتر، مدل فرصت پیدا می‌کند تا بهتر تعمیم دهد و خطای اعتبارسنجی به تدریج به خطای آموزش نزدیک شود.

موازنه‌ی بایاس و واریانس

در آمار و یادگیری ماشین، یک نکته‌ی کلیدی این است که خطای تعمیم یک مدل را می‌توان به سه جزء متمایز تقسیم کرد:

بایاس (Bias یا جهت‌گیری)

این بخش از خطا ناشی از فرضیات اشتباه مدل است؛ برای مثال، فرض خطی بودن داده‌ها در حالی که رابطه واقعی درجه دوم است. مدل‌هایی با بایاس بالا معمولاً کم‌برازش دارند و قادر به تطبیق دقیق داده‌های آموزشی نیستند.

واریانس (Variance یا میزان پراکندگی)

ین بخش از خطا به دلیل حساسیت بیش از حد مدل به تغییرات کوچک در داده‌های آموزشی ایجاد می‌شود. مدل‌هایی با درجه آزادی بالا، مانند یک مدل چندجمله‌ای با درجه بالا، واریانس بالایی دارند و معمولاً داده‌های آموزشی را بیش‌برازش می‌کنند.

© 2025 Mohsen Zare. All rights reserved.

خطای غیرقابل کاهش(*Irreducible error*)

این بخش ناشی از نویز موجود در داده‌ها است. تنها راه کاهش این نوع خطا، پاکسازی داده‌ها است؛ برای مثال رفع خطاهای منابع داده مانند حسگرهای خراب یا شناسایی و حذف داده‌های پرت.

© 2025 Mohsen Zare. All rights reserved.

مدل‌های خطی منظم

همان‌طور که در فصل‌های ۱ و ۲ بررسی شد، یکی از راهکارهای مؤثر برای کاهش بیش‌برازش، منظم‌سازی مدل است؛ یعنی محدود کردن درجه آزادی مدل. هرچه مدل ساده‌تر باشد، توانایی آن در برازش بیش از حد داده‌های آموزشی کمتر خواهد بود.

برای مدل‌های چندجمله‌ای، یک روش ساده برای منظم‌سازی، کاهش درجه چندجمله‌ای است.

در مدل‌های خطی، منظم‌سازی معمولاً با محدود کردن وزن‌های مدل انجام می‌شود. در ادامه، به بررسی سه تکنیک محبوب منظم‌سازی می‌پردازیم که هر کدام به شیوه‌ای متفاوت وزن‌ها را محدود می‌کنند:

- رگرسیون ریج (Ridge Regression)
- رگرسیون لاسو (Lasso Regression)
- رگرسیون شبکه الاستیک (Elastic Net Regression)

© 2025 Mohsen Zare. All rights reserved.

رگرسیون ریج

«رگرسیون ریج ($Ridge\ regression$)»—با منظم‌سازی L_2—که به آن «منظم‌سازی تیخونوف ($Tikhonov\ regularization$)» نیز گفته می‌شود، یک نسخه‌ی منظم شده از رگرسیون خطی است. در این روش، یک «تِرم منظم‌سازی ($regularization\ term$)» برابر با $\frac{\alpha}{m}\sum_{i=1}^{n}\theta_i^2$ به تابع هزینه MSE اضافه می‌شود. این عبارت باعث می‌شود الگوریتم یادگیری نه تنها داده‌ها را برازش دهد، بلکه وزن‌های مدل را نیز تا حد امکان کوچک نگه دارد. توجه: عبارت منظم‌سازی فقط در طول آموزش به تابع هزینه اضافه می‌شود. پس از آموزش، برای ارزیابی عملکرد مدل باید از MSE یا RMSE بدون منظم‌سازی استفاده کنید.

پارامتر α میزان منظم‌سازی مدل را تعیین می‌کند:

- اگر $\alpha = 0$ باشد، رگرسیون ریج همان رگرسیون خطی معمولی است.
- اگر α بسیار بزرگ باشد، وزن‌های مدل به شدت کوچک می‌شوند و در نتیجه یک خط تقریباً صاف که از میانگین داده‌ها عبور می‌کند، به دست می‌آید.

معادله ۴-۸، تابع هزینه رگرسیون ریج را به صورت رسمی نشان می‌دهد.

معادله ۴-۸. تابع هزینه رگرسیون ریج

$$J(\theta) = MSE(\theta) + \frac{\alpha}{m}\sum_{i=1}^{n}\theta_i^2$$

توجه داشته باشید که بایاس θ_0 در عبارت منظم‌سازی لحاظ نمی‌شود (جمع از $i = 1$ شروع می‌شود، نه ۰). اگر $\boldsymbol{w}$ را بردار وزن‌های ویژگی (θ_1 تا θ_n) در نظر بگیریم، عبارت منظم‌سازی برابر است با $\alpha(\| w \|_2)^2 / m$ خواهد بود، که در آن $\| w \|_2$ نشان‌دهنده نُرم ℓ_2 بردار وزن است. برای

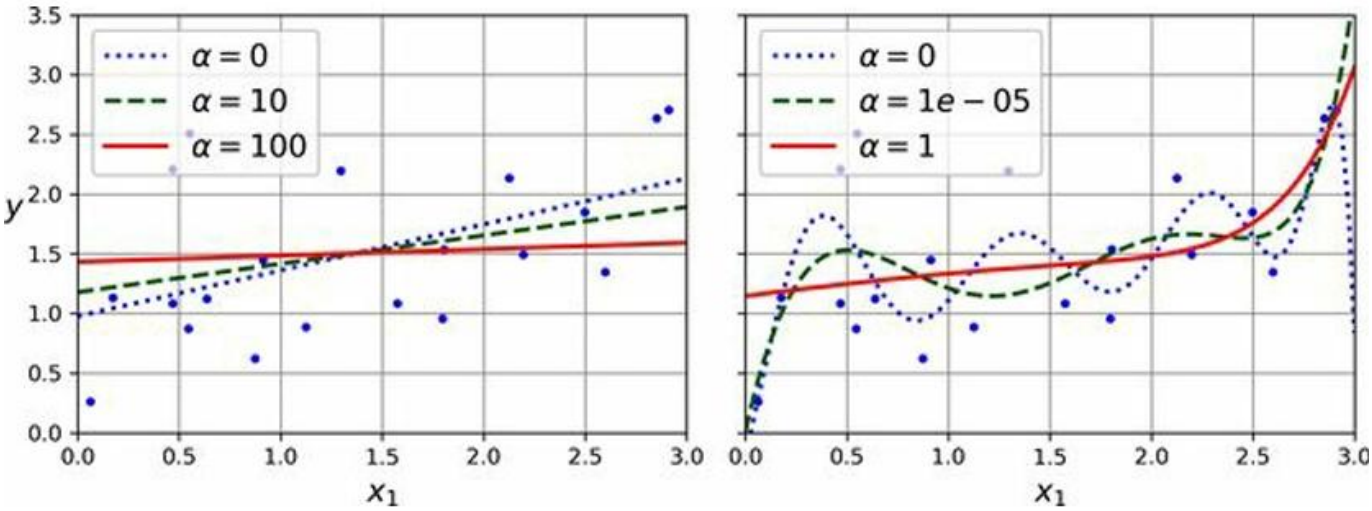
© 2025 Mohsen Zare. All rights reserved.

گرادیان نزولی دسته‌ای، کافی است $2\alpha \boldsymbol{w} / m$ را به بخش مربوط به وزن‌های ویژگی در بردار گرادیان MSE اضافه کنید، بدون اینکه تغییری در گرادیان بایاس اعمال شود (به معادله ۴-۶ مراجعه کنید).

هشدار

قبل از اجرای رگرسیون ریج، بسیار مهم است که داده‌ها را مقیاس‌بندی کنید (برای مثال با استفاده از StandardScaler)، زیرا این الگوریتم به مقیاس ویژگی‌های ورودی حساس است. این نکته در مورد اکثر مدل‌های منظم‌شده صادق است.

شکل ۴-۱۷ نمونه‌ای از چند مدل ریج را نشان می‌دهد که با مقادیر مختلف α روی داده‌های خطی بسیار نویزی آموزش داده شده‌اند. در سمت چپ، مدل‌های ریج ساده به کار رفته‌اند که پیش‌بینی‌های خطی ارائه می‌کنند. در سمت راست، ابتدا داده‌ها با PolynomialFeatures(degree=10) گسترش یافته، سپس با StandardScaler مقیاس‌بندی شده و در نهایت مدل‌های ریج روی ویژگی‌های حاصل اعمال شده‌اند؛ این همان رگرسیون چندجمله‌ای با منظم‌سازی ریج است. با افزایش α ، پیش‌بینی‌ها صاف‌تر می‌شوند؛ یعنی مقادیر افراطی کاهش یافته و مدل معقول‌تر عمل می‌کند. در نتیجه، واریانس مدل کاهش می‌یابد اما بایاس آن افزایش پیدا می‌کند.



شکل ۴-۱۷. مدل‌های خطی (چپ) و چندجمله‌ای (راست)، هر دو با سطوح مختلف منظم‌سازی ریج

© 2025 Mohsen Zare. All rights reserved.

همانند رگرسیون خطی، رگرسیون ریج را می‌توان یا با استفاده از معادله فرم بسته یا با گرادیان نزولی انجام داد. مزایا و معایب هر روش مشابه است. معادله ۴-۹ راه‌حل فرم بسته را نشان می‌دهد، که در آن $\mathbf{A}$ یک ماتریس همانی $(n+1) \times (n+1)$ است، با این تفاوت که در سلول بالای سمت چپ، مقدار صفر قرار دارد تا عبارت بایاس θ_0 منظم‌سازی نشود.

معادله ۴-۹. راه‌حل فرم بسته رگرسیون ریج

$$\hat{\theta} = (X^{\top} X + \alpha A)^{-1} \; X^{\top} y$$

در ادامه، روش اجرای رگرسیون ریج با Scikit-Learn از طریق یک راه‌حل فرم بسته نمایش داده شده است. این روش، نسخه‌ای از معادله ۴-۹ است که برای محاسبات از تکنیک فاکتورگیری ماتریسی Cholesky (André-Louis Cholesky) بهره می‌برد:

```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=0.1, solver="cholesky")
>>> ridge_reg.fit(X, y)
>>> ridge_reg.predict([[1.5]])
array([[1.55325833]])
```

و با استفاده از گرادیان نزولی تصادفی:

```
>>> sgd_reg = SGDRegressor(penalty="l2", alpha=0.1 / m, tol=None,
....                    max_iter=1000, eta0=0.01, random_state=42)
....
>>> sgd_reg.fit(X, y.ravel()) # y.ravel() because fit() expects 1D targets
>>> sgd_reg.predict([[1.5]])
array([1.55302613])
```

هایپرپارامتر penalty نوع عبارت منظم‌سازی مورد استفاده را مشخص می‌کند. اگر مقدار آن را "l2" قرار دهید، الگوریتم SGD یک عبارت منظم‌سازی به تابع هزینه MSE اضافه می‌کند که برابر با α ضربدر مربعِ نُرم ℓ_2 بردار وزن است. این دقیقاً مشابه رگرسیون ریج است، با این تفاوت که در این حالت تقسیم بر m انجام نمی‌شود. به همین دلیل، ما مقدار alpha = 0.1 / m را تنظیم کردیم تا نتیجه‌ای مشابه با Ridge(alpha=0.1) بدست آوریم.

© 2025 Mohsen Zare. All rights reserved.

نکته

کلاس RidgeCV نیز رگرسیون ریج را اجرا می‌کند، اما به‌طور خودکار هایپرپارامترها را با استفاده از اعتبارسنجی متقاطع تنظیم می‌کند. این تقریباً معادل استفاده از GridSearchCV است، اما برای رگرسیون ریج بهینه شده و سرعت اجرای آن بسیار بالاتر است. چندین تخمین‌گر خطی دیگر نیز نسخه‌های CV بهینه دارند، مانند LassoCV و ElasticNetCV.

© 2025 Mohsen Zare. All rights reserved.

رگرسیون Lasso

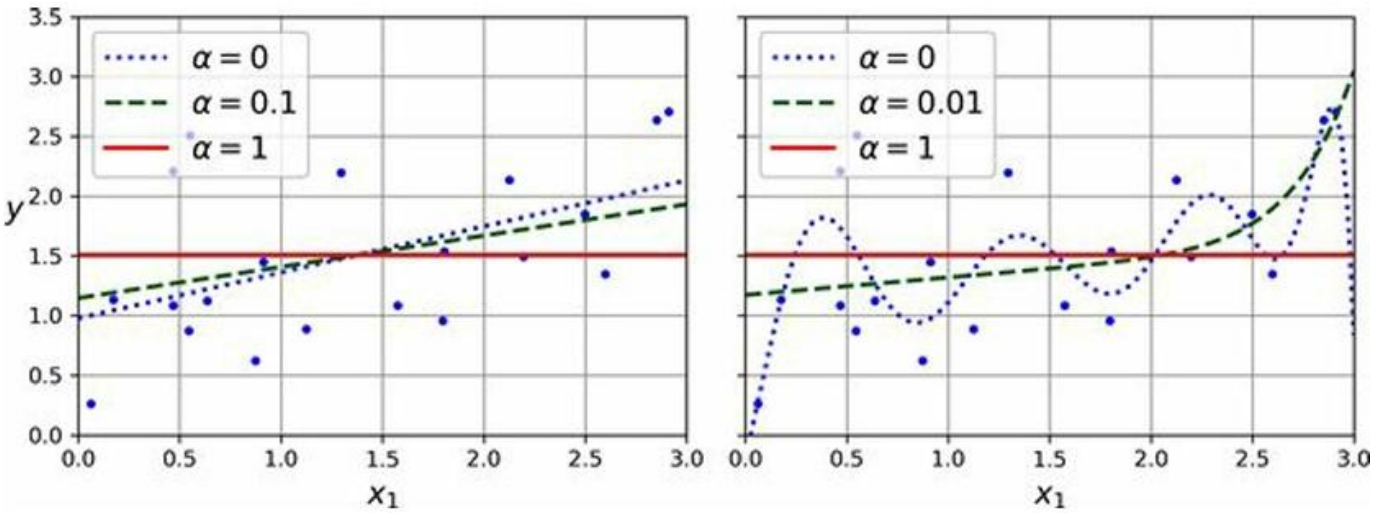
«رگرسیون کمترین کوچک‌سازی مطلق و انتخاب متغیرها (*Least absolute shrinkage and selection operator regression* که معمولاً برای سادگی رگرسیون LASSO (*lasso regression*) نامیده می‌شود)» نسخه‌ی منظم‌شده‌ی دیگری از رگرسیون خطی است. مشابه رگرسیون ریج، این روش یک عبارت منظم‌سازی به تابع هزینه اضافه می‌کند، اما به جای استفاده از مربع نُرم ℓ_2 بردار وزن، از نُرم ℓ_1 استفاده می‌کند (به معادله ۴-۱۰ مراجعه کنید). توجه داشته باشید که در لاسو، نُرم ℓ_1 در 2α ضرب می‌شود، در حالی که در رگرسیون ریج، نُرم ℓ_2 در α/m ضرب می‌شود. این ضرایب (factor) برای اطمینان از این انتخاب شده‌اند که مقدار بهینه‌ی α مستقل از اندازه‌ی مجموعه آموزشی باشد؛ زیرا نُرم‌های مختلف به فاکتورهای متفاوتی نیاز دارند (برای جزئیات بیشتر به شماره Scikit-Learn #15657 مراجعه کنید).

معادله ۴-۱۰. تابع هزینه رگرسیون لاسو

$$\mathrm{J}(\theta) = \mathrm{MSE}(\theta) + 2\alpha \sum_{i=1}^{n} \theta_i$$

شکل ۴-۱۸ مشابه شکل ۴-۱۷ است، با این تفاوت که در اینجا به جای مدل‌های ریج، از مدل‌های Lasso استفاده شده و مقادیر مختلف α نیز مورد بررسی قرار گرفته‌اند.

© 2025 Mohsen Zare. All rights reserved.



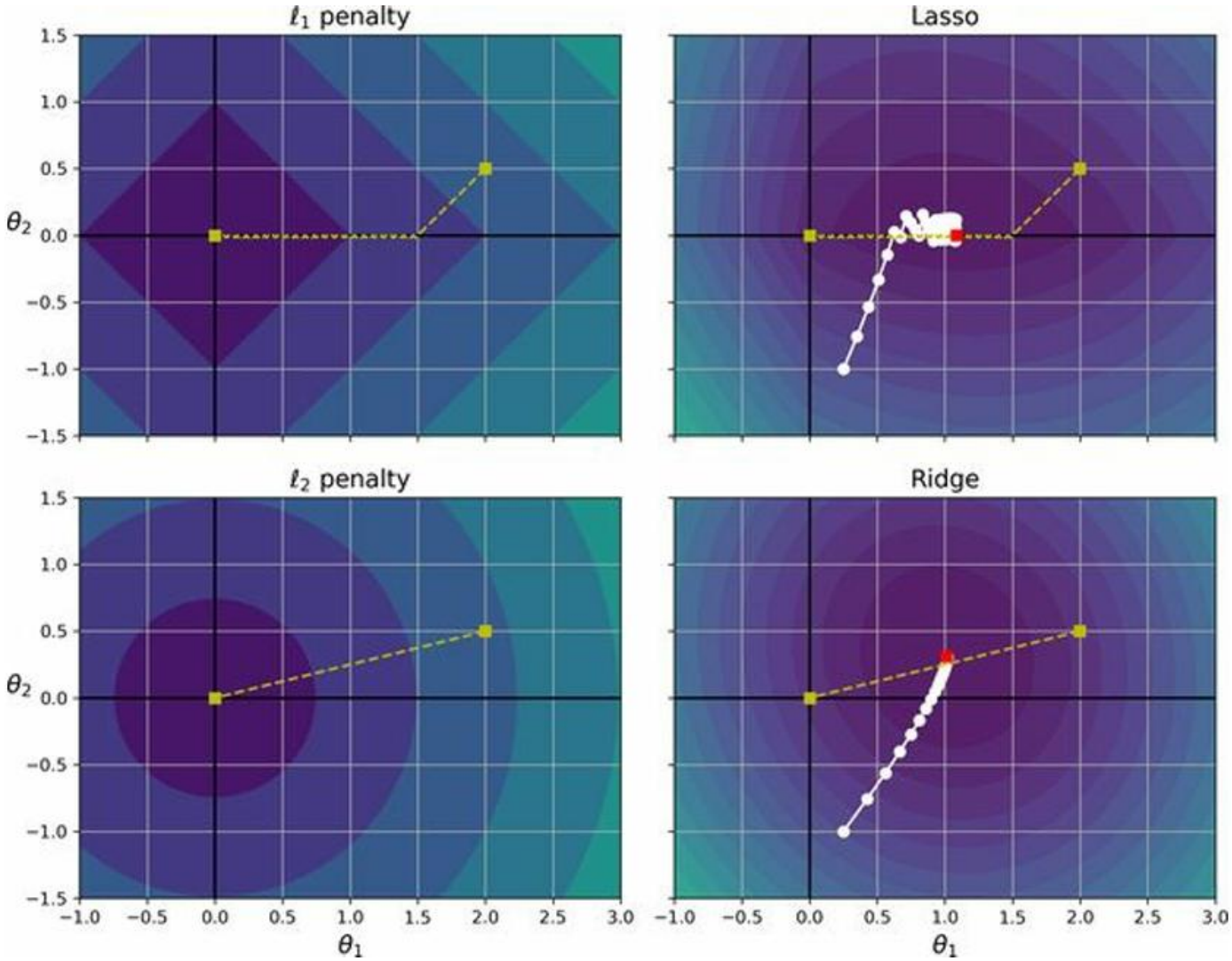
شکل ۴-۱۸. مدل‌های خطی (چپ) و چندجمله‌ای (راست)، هر دو با استفاده از سطوح مختلف منظم‌سازی لاسو

یکی از ویژگی‌های برجسته‌ی رگرسیون لاسو این است که تمایل دارد وزن ویژگی‌های کم‌اهمیت را به صفر برساند و عملاً آنها را حذف کند. برای مثال، در نمودار سمت راست شکل ۴-۱۸ (با $\alpha = 0.01$) تقریباً یک شکل مکعبی دیده می‌شود: تمام وزن‌های مربوط به ویژگی‌های چندجمله‌ای با درجه بالا برابر صفر هستند. به عبارت دیگر، رگرسیون لاسو به صورت خودکار انتخاب ویژگی انجام می‌دهد و یک «مدل تُنُک (*sparse model*)»—مدلی با پارامترها یا ضرایب عمدتاً صفر— با تعداد کمی وزن غیرصفر ایجاد می‌کند.

با نگاهی به شکل ۴-۱۹ می‌توان دلیل رفتار رگرسیون لاسو را فهمید: محورهای نمودار دو پارامتر مدل را نشان می‌دهند و خطوط پس‌زمینه، توابع زیان مختلف را نمایش می‌دهند. در نمودار بالا سمت چپ، خطوط نشان‌دهنده زیان(خطا) ℓ_1 برابر با $(|\theta_1| + |\theta_2|)$ هستند که با نزدیک شدن به هر محور، به صورت خطی کاهش می‌یابند. به عنوان مثال، اگر پارامترها با $\theta_1 = 2$ و $\theta_2 = 0.5$ مقداردهی اولیه کنید، گرادیان نزولی هر دو پارامتر را به طور همزمان کاهش می‌دهد (همان‌طور که با خط‌چین زرد نشان داده شده است)؛ بنابراین θ_2 ابتدا به صفر می‌رسد چون از ابتدا به صفر نزدیک‌تر بوده است. سپس گرادیان نزولی در امتداد ناودان(شیار یا کانال باریک)—مسیر باریکی در تابع هزینه LASSO که بعد از صفر شدن یک ضریب، گرادیان نزولی در آن سُر می‌خورد و ادامه می‌دهد—حرکت می‌کند تا θ_1 نیز به صفر برسد (با کمی جهش، زیرا گرادیان‌های ℓ_1 هرگز دقیقاً صفر نمی‌شوند و برای هر پارامتر یا ۱- یا ۱+ هستند). در نمودار بالا سمت راست، خطوط کانتور(contour)—خطوط هم‌تراز تابع هزینه؛ منحنی‌هایی که نقاط با مقدار ثابت تابع هزینه را نشان می‌دهند—نشان دهنده تابع هزینه رگرسیون لاسو هستند (یعنی یک تابع هزینه MSE به علاوه یک زیان ℓ_1). دایره‌های سفید کوچک مسیر گرادیان

© 2025 Mohsen Zare. All rights reserved.

نزولی برای بهینه‌سازی پارامترهایی که با $\theta_1 = 0.25$ و $\theta_2 = -1$ مقداردهی اولیه شده‌اند، نمایش می‌دهند. همان‌طور که مشاهده می‌کنید، مسیر به سرعت $\theta_2 = 0$ می‌رسد، سپس در امتداد حاشیه حرکت کرده و در نهایت به بهینه سراسری (مربع قرمز) می‌رسد. اگر α افزایش یابد، بهینه سراسری در امتداد خط‌چین زرد به سمت چپ حرکت می‌کند، و اگر α کاهش یابد، بهینه سراسری به سمت راست حرکت می‌کند. در این مثال، پارامترهای بهینه برای MSE نامنظم $\theta_1 = 2$ و $\theta_2 = 0.5$ هستند).



شکل ۴-۱۹. منظم‌سازی لاسو در مقابل منظم‌سازی ریج

دو نمودار پایینی همان موضوع را نشان می‌دهند، اما این بار با استفاده از جریمه ℓ_2. در نمودار پایین سمت چپ، مشاهده می‌کنید که با نزدیک شدن به مبدا، زیان ℓ_2 کاهش می‌یابد، بنابراین گرادیان نزولی مسیر مستقیمی به سمت نقطه بهینه طی می‌کند. در نمودار پایین سمت راست، خطوط کانتور

© 2025 Mohsen Zare. All rights reserved.

نمایانگر تابع هزینه رگرسیون ریج هستند (MSE به علاوه زیان ℓ_2). همان‌طور که می‌بینید، با نزدیک شدن پارامترها به بهینه سراسری، گرادیان‌ها کاهش می‌یابند و در نتیجه گرادیان نزولی به طور طبیعی کند می‌شود. این ویژگی، پرش‌های تصادفی را محدود می‌کند و به همگرایی سریع‌تر ریج نسبت به رگرسیون لاسو کمک می‌کند. همچنین توجه داشته باشید که پارامترهای بهینه (مربع قرمز) با افزایش α به تدریج به مبدا نزدیک‌تر می‌شوند، اما هرگز به طور کامل صفر نمی‌گردند.

نکته

برای کاهش پرش‌های گرادیان کاهشی در حوالی نقطه بهینه هنگام استفاده از رگرسیون لاسو، لازم است که در طول آموزش نرخ یادگیری را به تدریج کاهش دهید. با این کار، گرچه الگوریتم همچنان کمی در اطراف نقطه بهینه نوسان خواهد کرد، اما اندازه گام‌ها به تدریج کوچک می‌شوند و در نهایت الگوریتم همگرا می‌شود.

تابع هزینه لاسو در نقاطی که $\theta_i = 0$ برای $i = 1, 2, \cdots, n$ برقرار است، مشتق‌پذیر نیست. با این حال، با استفاده از یک «بردار زیرگرادیان $\mathbf{g}$ ($subgradient\ vector\ \mathbf{g}$)»—تعمیم گرادیان برای توابع غیرقابل مشتق (مثلاً L1-norm)، که جهت نزولی را نشان می‌دهد—می‌توان گرادیان نزولی را حتی در این نقاط به کار برد. معادله ۴-۱۱ نمونه‌ای از معادله برداری زیرگرادیان است که برای اجرای گرادیان نزولی با تابع هزینه لاسو قابل استفاده است.

معادله ۴-۱۱. بردار زیرگرادیان رگرسیون لاسو

$$\mathrm{g}(\theta, \mathrm{J}) = \nabla_\theta \mathrm{MSE}(\theta) + 2\alpha sign(\theta_1) sign(\theta_2) \vdots sign(\theta_n)$$

$$where\ sign(\theta_i) = -1\ if\ \theta_i < 0, 0\ if\ \theta_i = 0, +1\ if\ \theta_i > 0$$

$$g(\theta, J) = \nabla_\theta MSE(\theta) + 2\alpha \begin{bmatrix} sign(\theta_1) \\ sign(\theta_2) \\ \vdots \\ sign(\theta_n) \end{bmatrix} \quad , \quad sign(\theta_i) = \begin{cases} -1\ if & \theta_i < 0 \\ 0\ \ if & \theta_i = 0 \\ +1\ if & \theta_i > 0 \end{cases}$$

© 2025 Mohsen Zare. All rights reserved.

در ادامه یک مثال کوتاه از استفاده کلاس Lasso در Scikit-Learn ارائه شده است:

```
>>> from sklearn.linear_model import Lasso
>>> lasso_reg = Lasso(alpha=0.1)
>>> lasso_reg.fit(X, y)
>>> lasso_reg.predict([[1.5]])
array([1.53788174])
```

توجه داشته باشید که به جای استفاده از کلاس Lasso، می‌توانید از کلاس رگرسیون SGD به صورتِ SGDRegressor(penalty="l1", alpha=0.1) نیز استفاده کنید.

© 2025 Mohsen Zare. All rights reserved.

رگرسیون شبکه الاستیک

«رگرسیون شبکه الاستیک (*Elastic net regression*)»، ترکیبی از رگرسیون ریج و رگرسیون لاسو—که هم کوچک‌سازی مطلق (L_1) و هم جریمه مربعی (L_2) را به‌کار می‌برد—است. در این روش، عبارت منظم‌سازی، مجموع وزنی عبارات ریج و لاسو است و شما می‌توانید نسبت(ratio) ترکیب را با پارامتر r کنترل کنید. زمانی که $r = 0$ باشد، شبکه‌ی الاستیک معادل رگرسیون ریج است و زمانی که $r = 1$ باشد، معادل رگرسیون لاسو می‌شود(معادله ۴-۱۲).

معادله ۴-۱۲. تابع هزینه شبکه ی الاستیک

$$J(\theta) = MSE(\theta) + r2\alpha \sum_{i=1}^{n} \theta_i + (1-r)\frac{\alpha}{m}\sum_{i=1}^{n} \theta_i^2$$

بنابراین چه زمانی باید از رگرسیون شبکه الاستیک، ریج، لاسو یا رگرسیون خطی ساده (بدون منظم‌سازی) استفاده کرد؟ به طور کلی، توصیه می‌شود همیشه مقداری منظم‌سازی اعمال شود، بنابراین رگرسیون خطی ساده معمولاً انتخاب مناسبی نیست. رگرسیون ریج معمولاً به‌عنوان گزینه پیش‌فرض در نظر گرفته می‌شود. اگر فرض کنید تنها تعداد کمی از ویژگی‌ها واقعاً مفید هستند، بهتر است از لاسو یا شبکه الاستیک استفاده کنید، زیرا همان‌طور که توضیح داده شد، این دو روش وزن ویژگی‌های کم‌اهمیت را به صفر کاهش می‌دهند. در کل، شبکه الاستیک نسبت به لاسو انتخاب ایمن‌تری است، زیرا لاسو ممکن است در شرایطی که تعداد ویژگی‌ها بیشتر از تعداد نمونه‌ها باشد یا زمانی که ویژگی‌ها همبستگی بالایی با یکدیگر دارند، رفتار ناپایداری نشان دهد.

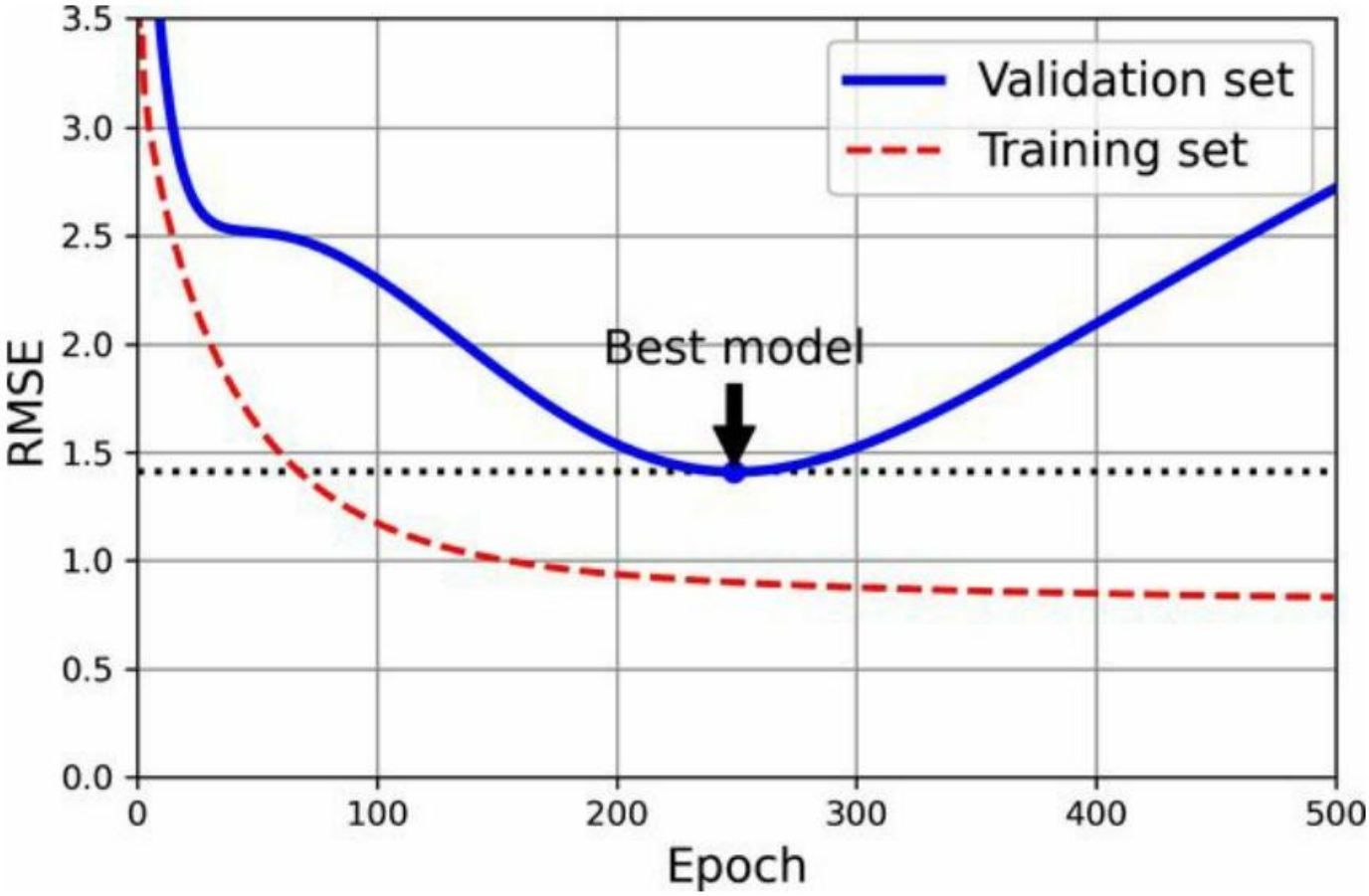
در اینجا یک نمونه‌ی ساده از استفاده‌ی کلاس ElasticNet در کتابخانه‌ی Scikit-Learn آورده شده است (که در آن پارامتر l1_ratio همان نسبت ترکیب r است):

```
>>> from sklearn.linear_model import ElasticNet
>>> elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
>>> elastic_net.fit(X, y)
>>> elastic_net.predict([[1.5]])
array([1.54333232])
```

© 2025 Mohsen Zare. All rights reserved.

توقف زودهنگام

روشی متفاوت برای منظم‌سازی الگوریتم‌های یادگیری تکراری مانند گرادیان نزولی این است که آموزش را درست در لحظه‌ای متوقف کنیم که خطای اعتبارسنجی به حداقل خود می‌رسد. به این روش، «توقف زودهنگام(*early stopping*)» گفته می‌شود. شکل ۴-۲۰ مدلی پیچیده (در اینجا یک رگرسیون چندجمله‌ای با درجه بالا) را نشان می‌دهد که با استفاده از گرادیان نزولی دسته‌ای روی همان داده‌های درجه دوم آموزش داده شده است. با گذشت دوره‌ها (epoch)، خطای پیش‌بینی مدل (RMSE) روی مجموعه آموزش و همچنین خطای آن روی مجموعه اعتبارسنجی کاهش می‌یابد. اما پس از مدتی، خطای اعتبارسنجی دیگر کاهش نمی‌یابد و دوباره افزایش پیدا می‌کند. این نقطه همان زمانی است که مدل شروع به بیش‌برازش داده‌های آموزش کرده است. در روش توقف زودهنگام، به محض رسیدن خطای اعتبارسنجی به حداقل مقدار خود، آموزش متوقف می‌شود. این تکنیک ساده و کارآمد، نوعی منظم‌سازی مؤثر به شمار می‌آید. جفری هینتون از آن با عنوان «ناهار رایگان زیبا» یاد کرده است.



شکل ۴-۲۰. منظم‌سازی با توقف زودهنگام

© 2025 Mohsen Zare. All rights reserved.

نکته

در روش‌های گرادیان کاهشی تصادفی (SGD) و مینی‌دسته‌ای، منحنی تغییرات خطا معمولاً نوسان دارد و به‌راحتی نمی‌توان تشخیص داد که آیا به نقطه مینیمم رسیده‌ایم یا نه. برای حل این مشکل، می‌توان از راهکاری استفاده کرد که در آن آموزش تنها زمانی متوقف شود که خطای اعتبارسنجی برای مدتی متوالی بالاتر از کمترین مقدار ثبت‌شده باقی بماند. در این صورت، اطمینان حاصل می‌شود که مدل دیگر بهبود نخواهد یافت. سپس کافی است پارامترهای مدل را به همان وضعیتی برگردانید که خطای اعتبارسنجی در کمترین مقدار خود قرار داشته است.

یک پیاده‌سازی ساده از توقف زودهنگام در Scikit-Learn و Numpy می‌تواند به شکل زیر باشد:

```
from copy import deepcopy
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

X_train, y_train, X_valid, y_valid = [...] # split the quadratic dataset

preprocessing = make_pipeline(PolynomialFeatures(degree=90, include_bias=False),
                              StandardScaler())
X_train_prep = preprocessing.fit_transform(X_train)
X_valid_prep = preprocessing.transform(X_valid)
sgd_reg = SGDRegressor(penalty=None, eta0=0.002, random_state=42)
n_epochs = 500
best_valid_rmse = float('inf')

for epoch in range(n_epochs):
    sgd_reg.partial_fit(X_train_prep, y_train)
    y_valid_predict = sgd_reg.predict(X_valid_prep)
    val_error = mean_squared_error(y_valid, y_valid_predict, squared=False)
  if val_error < best_valid_rmse:
      best_valid_rmse = val_error
      best_model = deepcopy(sgd_reg)
```

© 2025 Mohsen Zare. All rights reserved.

این کد ابتدا ویژگی‌های چندجمله‌ای را به داده‌ها اضافه می‌کند و سپس همه‌ی ویژگی‌های ورودی را، هم در مجموعه‌ی آموزش و هم در مجموعه‌ی اعتبارسنجی، مقیاس‌بندی می‌کند (با این فرض که مجموعه‌ی آموزشی اصلی را از پیش به دو بخش آموزش و اعتبارسنجی تقسیم کرده‌اید). در ادامه، یک مدل SGDRegressor با نرخ یادگیری پایین و بدون هیچ‌گونه منظم‌سازی ساخته می‌شود. در حلقه‌ی آموزش، به جای استفاده از متد ()fit، از متد ()partial_fit بهره گرفته می‌شود تا فرایند یادگیری به‌صورت افزایشی انجام شود. در هر دوره، مقدار RMSE روی داده‌ی اعتبارسنجی محاسبه می‌گردد. اگر این مقدار کمتر از بهترین RMSE مشاهده‌شده تاکنون باشد، یک نسخه‌ی کامل از مدل در متغیر best_model ذخیره می‌شود. این پیاده‌سازی در عمل آموزش را متوقف نمی‌کند، اما به شما امکان می‌دهد پس از پایان آموزش به بهترین مدل بازگردید. توجه داشته باشید که برای کپی گرفتن از مدل از ()copy.deepcopy استفاده می‌شود، زیرا این روش هم ساختار مدل «و هم» پارامترهای آموخته‌شده را کپی می‌کند، در حالی که متد ()sklearn.base.clone تنها پارامترهای مدل را کپی می‌کند.

© 2025 Mohsen Zare. All rights reserved.

رگرسیون لجستیک

همان‌طور که در فصل ۱ اشاره شد، برخی الگوریتم‌های رگرسیون می‌توانند برای طبقه‌بندی نیز به کار روند (و بالعکس). «رگرسیون لجستیک (*Logistic regression*)»–که همچنین رگرسیون لوجیت (*logit regression*)» نامیده می‌شود–عمدتاً برای برآورد احتمال تعلق یک نمونه به یک کلاس مشخص استفاده می‌شود. به عنوان مثال، می‌توان با آن احتمال هرزنامه بودن یک ایمیل را تخمین زد. اگر این احتمال از یک آستانه‌ی معین (به طور معمول ۵۰%) بیشتر باشد، مدل پیش‌بینی می‌کند که نمونه متعلق به «کلاس مثبت (*Positive Class*)» است که با برچسب «۱» نمایش داده می‌شود. در غیر این صورت، مدل آن را در «کلاس منفی (*Negative Class*)» قرار می‌دهد که با برچسب «۰» مشخص می‌شود. به این ترتیب، رگرسیون لجستیک یک «طبقه‌بند باینری (*Binary Classifier*)» به شمار می‌آید.

© 2025 Mohsen Zare. All rights reserved.

تخمین احتمالات

رگرسیون لجستیک چگونه کار می‌کند؟ مانند یک مدل رگرسیون خطی، رگرسیون لجستیک نیز مجموع وزنی ویژگی‌های ورودی (به‌علاوه یک عبارت بایاس) را محاسبه می‌کند. با این تفاوت که به جای ارائه‌ی مستقیم نتیجه، همان‌طور که در رگرسیون خطی انجام می‌شود، خروجی را به صورت احتمال مبتنی بر « تابع لجستیک ($logistic$)» ارائه می‌دهد. برای جزئیات ریاضی می‌توانید به معادله ۴-۱۳ مراجعه کنید.

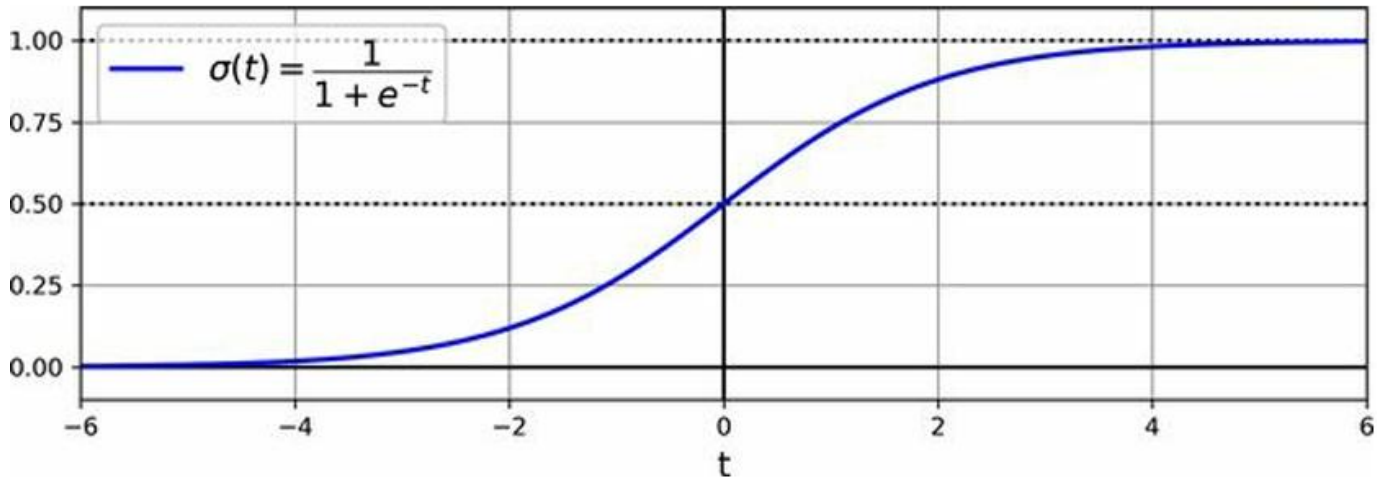
معادله ۴-۱۳. احتمال تخمینی مدل رگرسیون لجستیک (فرم برداری)

$$\hat{p} = h_\theta(x) = \sigma(\theta^\top x)$$

تابع لجستیک، که با $\sigma(\cdot)$ نمایش داده می‌شود، یک تابع سیگموئید (S شکل) است که خروجی آن همواره بین ۰ و ۱ قرار دارد. این تابع در معادله ۴-۱۴ و شکل ۴-۲۱ به طور دقیق تعریف شده است.

معادله ۴-۱۴. تابع لجستیک

$$\sigma(t) = \frac{1}{1 + exp(-t)}$$



شکل ۴-۲۱. تابع لجستیک

© 2025 Mohsen Zare. All rights reserved.

زمانی که مدل رگرسیون لجستیک احتمال $\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x})$ را برای تعلق یک نمونه $\mathbf{x}$ به کلاس مثبت تخمین زد، می‌تواند به سادگی پیش‌بینی $\hat{y}$ خود را انجام دهد (به معادله ۴-۱۵ مراجعه کنید).

معادله ۴-۱۵. پیش‌بینی مدل رگرسیون لجستیک با استفاده از احتمال آستانه ۵۰%

$$\hat{y} = \begin{cases} 0 & if \quad \hat{p} < 0.5 \\ 1 & if \quad \hat{p} \geq 0.5 \end{cases}$$

توجه داشته باشید که وقتی $t < 0$، $\sigma(t) < 0.5$ است و وقتی $t \geq 0$، $\sigma(t) \geq 0.5$ خواهد بود، بنابراین، با استفاده از آستانه پیش‌فرض ۵۰%، مدل رگرسیون لجستیک مقدار ۱ را پیش‌بینی می‌کند اگر $\boldsymbol{\theta}^{\top}\mathbf{x}$ مثبت باشد و مقدار ۰ را پیش‌بینی می‌کند اگر منفی باشد.

نکته

امتیاز t معمولاً «لوجیت ($logit$)» نامیده می‌شود. این نام از آنجا می‌آید که تابع لوجیت، که به صورت $logit(p) = log(p/(1-p))$ عریف می‌شود، در واقع معکوس تابع لجستیک است. به بیان دیگر، اگر لوجیتِ احتمال تخمینی p را محاسبه کنید، دقیقاً به همان مقدار t می‌رسید. همچنین لوجیت را «لگاریتم شانس ($log\text{-}odds$)» نیز می‌نامند، زیرا برابر است با لگاریتم نسبت احتمال تعلق نمونه به کلاس مثبت بر احتمال تعلق آن به کلاس منفی– لگاریتم نسبت احتمال موفقیت به احتمال شکست.

© 2025 Mohsen Zare. All rights reserved.

آموزش و تابع هزینه

اکنون که می‌دانید یک مدل رگرسیون لجستیک چگونه احتمال‌ها را تخمین می‌زند و پیش‌بینی می‌کند، پرسش اصلی این است که چگونه آموزش داده می‌شود؟ هدف آموزش، یافتن بردار پارامتر $\boldsymbol{\theta}$ است به گونه‌ای که مدل برای نمونه‌های مثبت $(y = 1)$ احتمال‌های بزرگ و برای نمونه‌های منفی $(y = 0)$ احتمال‌های کوچک تخمین بزند. این مفهوم با تابع هزینه بیان می‌شود که در معادله‌ی (۴-۱۶) برای یک نمونه آموزشی منفرد $\mathbf{x}$ تعریف شده است.

معادله ۴-۱۶. تابعِ هزینه‌ی یک نمونه آموزشی واحد

$$c(\theta) = \begin{cases} -\log(\hat{p}) & if \quad y = 1 \\ -\log(1 - \hat{p}) & if \quad y = 0 \end{cases}$$

این تابع هزینه منطقی است، زیرا $-log(t)$ هنگامی که t به صفر نزدیک می‌شود، مقدار بسیار بزرگی پیدا می‌کند. بنابراین اگر مدل برای یک نمونه مثبت احتمالی نزدیک به صفر پیش‌بینی کند، هزینه بسیار زیاد خواهد بود. به همین ترتیب، اگر برای یک نمونه منفی احتمالی نزدیک به ۱ برآورد شود، هزینه باز هم بزرگ خواهد بود. در مقابل، زمانی که t به ۱ نزدیک باشد، $-log(t)$ به صفر میل می‌کند. به همین دلیل، اگر مدل برای یک نمونه مثبت احتمالی نزدیک به یک و برای یک نمونه منفی احتمالی نزدیک به صفر پیش‌بینی کند، هزینه نزدیک به صفر خواهد بود. این دقیقاً رفتاری است که در فرایند آموزش انتظار داریم.

تابع هزینه کل برای مجموعه آموزشی، برابر با میانگین هزینه روی تمام نمونه‌ها است. این تابع معمولاً در قالب یک عبارت فشرده به نام «لگاریتم زیان(*log loss*)» یا همان Binary Cross-Entropy بیان می‌شود که در معادله ۴-۱۷ نشان داده شده است.

© 2025 Mohsen Zare. All rights reserved.

معادله ۴-۱۷. تابع هزینه رگرسیون لجستیک (لگاریتم زیان)

$$J(\theta) = -1m\sum_{i=1}^{m}\left[y^{(i)}log\hat{\mathrm{p}}^{(i)} + \left(1 - y^{(i)}\right)log1 - \hat{\mathrm{p}}^{(i)}\right]$$

هشدار

لگاریتم زیان به صورت تصادفی یا بدون دلیل انتخاب نشده است. می‌توان نشان داد که به حداقل رساندن این تابع هزینه، از دیدگاه ریاضی و با استدلال بِیزیَن، معادل با به‌دست آوردن مدلی است که «برآوردِ ماکسیمُم درست‌نمایی (*Maximum Likelihood Estimation* – MLE)»–روشی استاندارد برای تَخمین پارامترها با بیشینه‌کردنِ احتمالِ مشاهده‌ی داده‌ها–را محقق می‌کند؛ به شرط آنکه فرض کنیم نمونه‌ها از یک توزیع گاوسی پیرامون میانگین کلاس خود پیروی می‌کنند. بنابراین، استفاده از لگاریتم زیان به‌طور ضمنی همین فرض را در بر دارد. هرچه این فرض کمتر با واقعیت داده‌ها تطابق داشته باشد، مدل دچار بایاسِ بیشتری خواهد شد. به طور مشابه، زمانی که از MSE برای آموزش مدل‌های رگرسیون خطی استفاده کردیم، در واقع فرض ضمنی ما این بود که رابطه داده‌ها کاملاً خطی است و تنها مقداری نویز با توزیع گاوسی وجود دارد. اگر داده‌ها واقعاً خطی نباشند (مثلاً دارای الگوی درجه دوم باشند) یا نویز از نوع گاوسی نباشد (مثلاً داده‌های پرت به صورت نمایی یا نادر وجود داشته باشند)، در این صورت مدل دچار بایاس خواهد شد.

خبر ناخوشایند این است که برای این تابع هزینه، هیچ معادله راه حل بسته (closed-form solution)–راه‌حل به‌دست‌آمده با استفاده از روابط ریاضی بسته (بدون نیاز به تکرار یا تقریب)– شناخته‌شده‌ای برای به‌دست آوردن مقادیر $\boldsymbol{\theta}$ وجود ندارد (برخلاف معادله نرمال در رگرسیون خطی). اما خبر خوش این است که این تابع هزینه محدب (convex) است؛ بنابراین با استفاده از گرادیان نزولی یا هر الگوریتم بهینه‌سازی دیگر، تضمین می‌شود که به مینیمم سراسری (global minimum) برسیم؛ به شرطی که نرخ یادگیری بیش از حد بزرگ نباشد و تعداد تکرارها کافی باشد. مشتق جزئی تابع هزینه نسبت به پارامتر مدلِ j اُم، از θ_j، توسط معادله ۴-۱۸ ارائه شده است.

© 2025 Mohsen Zare. All rights reserved.

معادله ۴-۱۸. مشتقات جزئی تابع هزینه لجستیک

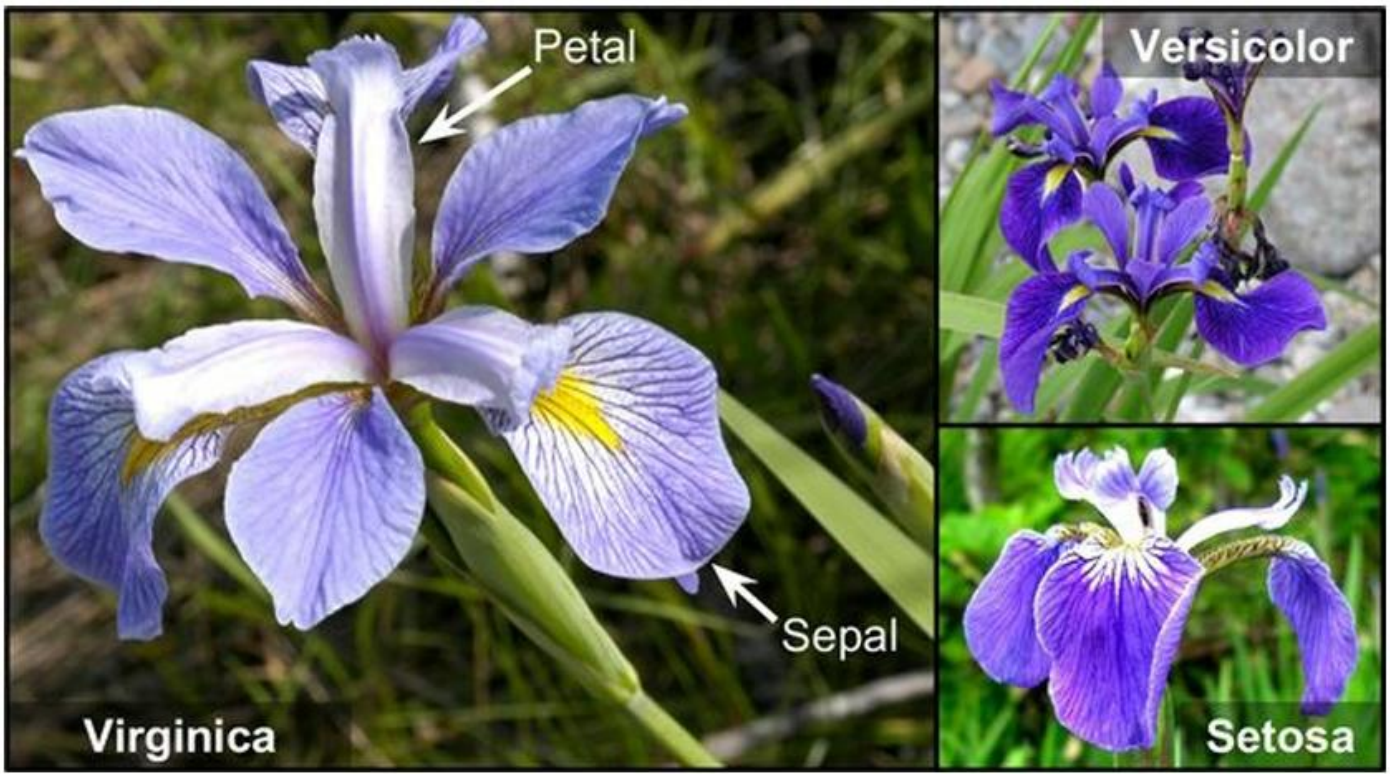
$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m} i = \frac{1}{m}\,\sigma\left(\theta^\top x^{(i)}\right) - y^{(i)}\,x_j^{(i)} \quad \text{for} \quad j = 0, \dots, n$$

این معادله شباهت زیادی به معادله ۴-۵ دارد: در هر نمونه ابتدا خطای پیش‌بینی محاسبه می‌شود، سپس در مقدار ویژگی j اُم ضرب می‌گردد و در نهایت میانگین این مقادیر برای تمام نمونه‌های آموزشی گرفته می‌شود. وقتی بردار گرادیان شامل تمام مشتقات جزئی را در اختیار داشته باشید، می‌توانید از آن در گرادیان نزولی دسته‌ای (Batch GD) استفاده کنید. به همین سادگی: حالا می‌دانید چگونه یک مدل رگرسیون لجستیک را آموزش دهید. در گرادیان نزولی تصادفی (SGD) هر بار تنها یک نمونه به‌روزرسانی می‌شود، و در گرادیان نزولی مینی‌دسته‌ای (Mini-batch GD) هر بار یک دسته‌ی کوچک از نمونه‌ها به کار گرفته می‌شود.

© 2025 Mohsen Zare. All rights reserved.

مرزهای تصمیم‌گیری

برای نشان دادن عملکرد رگرسیون لجستیک می‌توان از مجموعه‌داده‌ی معروف «گل زنبق (*Iris dataset*)» استفاده کرد. این مجموعه شامل ۱۵۰ نمونه از سه گونه‌ی مختلف زنبق است: *Iris setosa*، *Iris versicolor* و *Iris virginica*. برای هر نمونه، طول و عرض کاسبرگ(sepal) و گلبرگ(petal) اندازه‌گیری شده است (به شکل ۴-۲۲ مراجعه کنید).



شکل ۴-۲۲. گل‌های سه گونه گیاه زنبق

بیایید یک طبقه‌بند طراحی کنیم که بتواند گونه‌ی *Iris virginica* را تنها با استفاده از ویژگی عرض گلبرگ شناسایی کند. مرحله‌ی نخست، بارگذاری داده‌ها و بررسی اولیه آن‌ها است:

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris(as_frame=True)
list(iris)
['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module']
>>> iris.data.head(3)
```

© 2025 Mohsen Zare. All rights reserved.

```
   sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)
0               5.1               3.5                1.4               0.2
1               4.9               3.0                1.4               0.2
2               4.7               3.2                1.3               0.2
>>> iris.target.head(3) # note that the instances are not shuffled
0    0
1    0
2    0
Name: target, dtype: int64
>>> iris.target_names
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

در مرحله‌ی بعد، داده‌ها را به مجموعه‌های آموزشی و آزمایشی تقسیم کرده و مدل رگرسیون لجستیک را روی مجموعه آموزشی آموزش می‌دهیم:

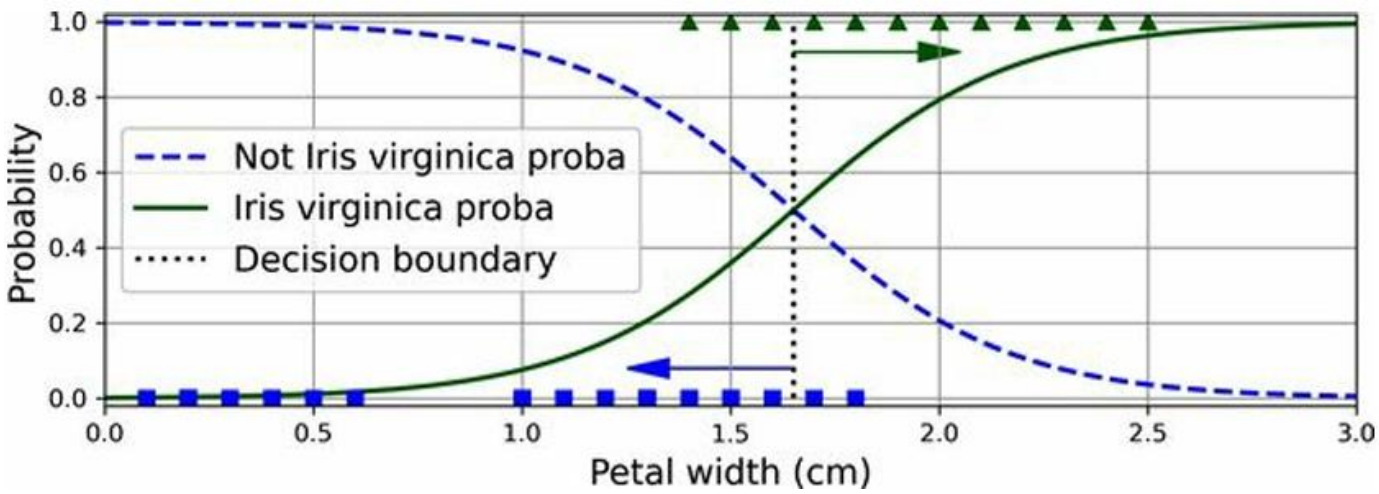
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X = iris.data[["petal width (cm)"]].values
y = iris.target_names[iris.target] == 'virginica'
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
```

حال بیایید احتمالات تخمینی مدل را برای گل‌هایی با عرض گلبرگ در بازه‌ی ۰ تا ۳ سانتی‌متر بررسی کنیم (شکل ۴-۲۳):

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1) # reshape to get a column vector
y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0, 0]

plt.plot(X_new, y_proba[:, 0], "b--", linewidth=2,
    label="Not Iris virginica proba")
plt.plot(X_new, y_proba[:, 1], "g-", linewidth=2, label="Iris virginica proba")
plt.plot([decision_boundary, decision_boundary], [0, 1], "k:", linewidth=2,
    label="Decision boundary")
```

© 2025 Mohsen Zare. All rights reserved.

```
[...] # beautify the figure: add grid, labels, axis, legend, arrows, and samples
plt.show()
```



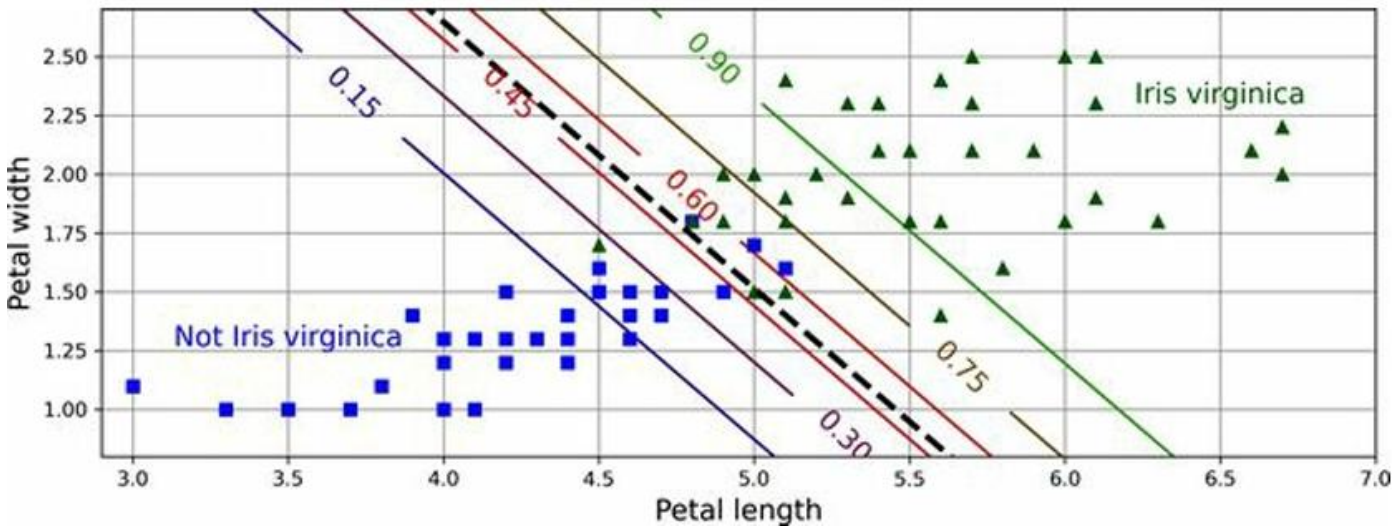
شکل ۴-۲۳. احتمالات تخمینی و مرز تصمیم‌گیری

عرض گلبرگ گل‌های زنبق ویرجینیا (*Iris virginica*)، که با مثلث نمایش داده شده‌اند، بین ۱٫۴ تا ۲٫۵ سانتی‌متر قرار دارد، در حالی که سایر گونه‌های زنبق (با مربع نمایش داده شده‌اند) معمولاً عرض گلبرگ کمتری دارند و در بازه ۰٫۱ تا ۱٫۸ سانتی‌متر هستند. البته کمی همپوشانی بین این دو گروه وجود دارد. برای عرض گلبرگ‌های بالاتر از حدود ۲ سانتی‌متر، طبقه‌بند با اطمینان بالایی پیش‌بینی می‌کند که گل متعلق به *Iris virginica* است و احتمال بالایی برای این دسته ارائه می‌دهد. در مقابل، برای عرض‌های کمتر از ۱ سانتی‌متر، با اطمینان پیش‌بینی می‌کند که گل متعلق به این گونه نیست. بین این دو حد افراطی، طبقه‌بند مطمئن نیست. با این حال، اگر از تابع predict() استفاده کنید به جای predict_proba()، کلاس محتمل‌تر برگردانده می‌شود. بنابراین یک «مرز تصمیم‌گیری (decision boundary)» تقریباً در عرض ۱٫۶ سانتی‌متر ایجاد می‌شود، جایی که هر دو احتمال برابر با ۵۰% هستند. اگر عرض گلبرگ بیشتر از ۱٫۶ سانتی‌متر باشد، مدل پیش‌بینی می‌کند که گل *Iris virginica* است و در غیر این صورت، پیش‌بینی می‌کند که نیست، حتی اگر اطمینان آن زیاد نباشد:

```
>>> decision_boundary
1.6516516516516517
>>> log_reg.predict([[1.7], [1.5]])
array([ True, False])
```

© 2025 Mohsen Zare. All rights reserved.

شکل ۴-۲۴ همان مجموعه داده را نشان می‌دهد، اما این بار هر گل با دو ویژگی نمایش داده شده است: طول و عرض گلبرگ. پس از آموزش، طبقه‌بند رگرسیون لجستیک می‌تواند بر اساس این دو ویژگی، احتمال تعلق یک گل جدید به گونه *Iris virginica* را تخمین بزند. خط چین، نقاطی را نشان می‌دهد که مدل در آن‌ها احتمال ۵۰% را تخمین می‌زند؛ این همان مرز تصمیم‌گیری مدل است. توجه داشته باشید که این مرز خطی است. همچنین خطوط موازی دیگری وجود دارند که احتمال‌های مختلف را نشان می‌دهند، از ۱۵% در پایین سمت چپ تا ۹۰% در بالا سمت راست. طبق مدل، تمام گل‌هایی که در بالای خط سمت راست قرار دارند، بیش از ۹۰% احتمال تعلق به گونه *Iris virginica* دارند.



شکل ۴-۲۴. مرز تصمیم‌گیری خطی

نکته

پارامتری که شدت منظم‌سازی در رگرسیون لجستیک Scikit-Learn را کنترل می‌کند، α نیست (مانند برخی مدل‌های خطی دیگر)، بلکه معکوس آن یعنی C است. مقدار بزرگ‌تر C به معنای منظم‌سازی کمتر و انعطاف‌پذیری بیشتر مدل است، در حالی که مقدار کوچک‌تر C منظم‌سازی قوی‌تر و محدودیت بیشتر روی وزن‌ها ایجاد می‌کند.

همانند سایر مدل‌های خطی، رگرسیون لجستیک را می‌توان با استفاده از جریمه‌های ℓ_1 یا ℓ_2 منظم‌سازی کرد. در Scikit-Learn، به‌طور پیش‌فرض از جریمه ℓ_2 استفاده می‌شود.

© 2025 Mohsen Zare. All rights reserved.

رگرسیون سافت‌مَکس(softmax)

می‌توان مدل رگرسیون لجستیک را طوری تعمیم داد که مستقیماً از چند کلاس پشتیبانی کند، بدون نیاز به آموزش و ترکیب چندین طبقه‌بند باینری (همان‌طور که در فصل ۳ بحث شد). این مدل، «رگرسیون سافت‌مَکس (*softmax regression*) یا رگرسیون لجستیک چندجمله‌ای (*multinomial logistic regression*)» نامیده می‌شود.

ایده‌ی اصلی ساده است: وقتی نمونه‌ای $\mathbf{x}$ داده می‌شود، مدل ابتدا برای هر کلاسِ k یک امتیاز $s_k(\mathbf{x})$ محاسبه می‌کند و سپس احتمال هر کلاس را با اعمال تابع سافت‌مَکس، که به آن «تابع نمایی نرمال‌شده (*normalized exponential*)» نیز گفته می‌شود، روی این امتیازها تخمین می‌زند. معادله محاسبه $s_k(\mathbf{x})$ مشابه پیش‌بینی در رگرسیون خطی است و به معادله ۴-۱۹ شبیه می‌باشد.

معادله ۴-۱۹. امتیاز سافت‌مکس برای کلاس k

$$s_k(x) = \left(\theta^{(k)}\right)^{\top} x$$

توجه داشته باشید که برای هر کلاس، یک بردار پارامتر مخصوص خود، $\boldsymbol{\theta}^{(k)}$، وجود دارد. معمولاً تمام این بردارها به‌صورت سطرهایی در «ماتریس پارامتر $\boldsymbol{\Theta}$» ذخیره می‌شوند.

پس از محاسبه امتیاز هر کلاس برای نمونه $\mathbf{x}$، می‌توان احتمال $\hat{p}_k$ که نمونه به کلاس k تعلق دارد، را با اعمال تابع سافت‌مکس (معادله ۴-۲۰) بر روی این امتیازات تخمین زد. این تابع، نمایی(توان) از هر امتیاز را محاسبه کرده و سپس آن‌ها را نرمال می‌کند (با تقسیم بر مجموع تمام نماها). این امتیازات معمولاً logits یا log-odds نامیده می‌شوند، اگرچه در واقع این‌ها log-oddsهای نرمال‌شده نیستند.

معادله ۴-۲۰. تابع سافت‌مکس

$$\hat{p}_k = \sigma\big(s(x)\big)_k = \frac{exp\ s_k(x)}{\sum_{j=1}^{K} exp\ s_j(x)} \qquad \text{for} \quad k = 1, \dots, K$$

© 2025 Mohsen Zare. All rights reserved.

در این معادله:

- K تعداد کلاس‌ها است.
- $\mathbf{s}(\mathbf{x})$ برداری شامل امتیازهای هر کلاس برای نمونه $\mathbf{x}$ است.
- $\boldsymbol{\sigma}\left(\mathbf{s}(\mathbf{x})\right)_k$ احتمال تخمینی تعلق نمونه $\mathbf{x}$ به کلاس k با توجه به امتیازهای هر کلاس برای آن نمونه را نشان می‌دهد.

درست مانند طبقه‌بندِ رگرسیون لجستیک، طبقه‌بند رگرسیون سافت‌مکس به‌طور پیش‌فرض، نمونه را به کلاسی اختصاص می‌دهد که بیشترین احتمال تخمینی را دارد؛ یعنی همان کلاسی که بالاترین امتیاز را دارد، همان‌طور که در معادله ۴-۲۱ نشان داده شده است.

معادله ۴-۲۱. پیش‌بینی طبقه‌بند رگرسیون سافت‌مکس

$$\hat{y} = argmax_k\ \sigma s(x)_k = argmax_k\ s_k(x) = argmax_k\ \left(\theta^{(k)}\right)^{\top} x$$

عملگر $argmax$ مقداری را برمی‌گرداند که یک تابع را بیشینه می‌کند. در این معادله، k را برمی‌گرداند که در آن احتمال تخمینی $\sigma\left(\mathbf{s}(\mathbf{x})\right)_k$ بیشترین مقدار را دارد.

نکته

طبقه‌بند رگرسیون سافت‌مکس تنها یک کلاس را برای هر نمونه پیش‌بینی می‌کند(یعنی چندکلاسه است، نه چندخروجی). بنابراین، این مدل مناسب مسائلی با کلاس‌های متقابل ناسازگار است، مانند گونه‌های مختلف گیاهان. نمی‌توان از آن برای مسائلی استفاده کرد که یک نمونه ممکن است همزمان به چند کلاس تعلق داشته باشد، مانند تشخیص چندین نفر در یک تصویر.

حالا که می‌دانید مدل چگونه احتمالات را تخمین می‌زند و پیش‌بینی می‌کند، بیایید به آموزش آن

© 2025 Mohsen Zare. All rights reserved.

بپردازیم. هدف، تولید مدلی است که برای کلاس هدف احتمال بالا و برای سایر کلاس‌ها احتمال پایین تخمین بزند. مینیمم‌سازی تابع هزینه نشان داده شده در معادله ۴-۲۲، که به آن «آنتروپی متقاطع ($cross\ entropy$)»—معیار اندازه‌گیری اختلاف بین دو توزیع احتمالات: توزیع واقعی و توزیع پیش‌بینی‌شده مدل—گفته می‌شود، دقیقاً به این هدف کمک می‌کند، زیرا اگر مدل احتمال کمی برای کلاس هدف تخمین بزند، جریمه می‌شود. آنتروپی متقاطع اغلب برای سنجش میزان تطابق مجموعه‌ای از احتمالات کلاس تخمینی با کلاس‌های هدف واقعی به کار می‌رود.

معادله ۴-۲۲. تابع هزینه آنتروپی متقاطع

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)}\ log\hat{p}_k^{(i)}$$

در این معادله، $y_k^{(i)}$ نشان‌دهنده احتمال هدفِ تعلقِ نمونه i اُم به کلاس k است که به طور معمول برابر ۱ یا ۰ است، بسته به اینکه نمونه به آن کلاس تعلق داشته باشد یا خیر.

توجه داشته باشید که وقتی تنها دو کلاس وجود دارد ($K = 2$)، این تابع هزینه دقیقاً معادل تابع هزینه رگرسیون لجستیک (لگاریتم زیان یا $\log$ loss؛ به معادله ۴-۱۷ مراجعه کنید) خواهد بود.

© 2025 Mohsen Zare. All rights reserved.

آنتروپی متقاطع

آنتروپی متقاطع (cross-entropy) ریشه در «نظریه‌ی. اطلاعات» کلود شانون دارد. فرض کنید می‌خواهید هر روز اطلاعات آب و هوا را به شکل کارآمد منتقل کنید. اگر هشت حالت ممکن وجود داشته باشد (مثلاً آفتابی، بارانی و غیره)، می‌توان هر حالت را با ۳ بیت کدگذاری کرد، زیرا $2^3 = 8$. با این حال، اگر بدانید که تقریباً هر روز آفتابی است، می‌توانید «آفتابی» را تنها با یک بیت (۰) و هفت گزینه دیگر را با چهار بیت (با شروع از ۱) کدگذاری کنید که بسیار کارآمدتر است. آنتروپی متقاطع میانگین تعداد بیت‌هایی را اندازه‌گیری می‌کند که واقعاً برای انتقال هر گزینه ارسال می‌کنید. اگر فرض شما درباره آب و هوا دقیق باشد، آنتروپی متقاطع برابر با آنتروپی واقعی آب و هوا خواهد بود (یعنی میزان غیرقابل پیش‌بینی بودن ذاتی آن). اما اگر فرض شما نادرست باشد (مثلاً فکر کنید اغلب باران می‌بارد)، آنتروپی متقاطع بزرگ‌تر خواهد شد و تفاوت آن با آنتروپی واقعی با واگرایی کولبک-لیبلر ($Kullback–Leibler\ (KL)\ divergence$) سنجیده می‌شود.

آنتروپی متقاطع بین دو توزیع احتمال p و q به صورت $H(p,q) =- \Sigma_x p(x)\ log q(x)$ است. که در آن: $p(x)$ ← توزیع واقعی داده‌ها، $q(x)$ ← توزیع پیش‌بینی‌شده مدل و مجموع روی همه حالت‌های ممکن x انجام می‌شود. این تعریف برای توزیع‌های گسسته صادق است. برای درک عمیق‌تر و توضیحات تصویری، می‌توانید ویدیوی مرتبط من در این مورد را نیز مشاهده کنید.

بردار گرادیان تابع هزینه آنتروپی متقاطع نسبت به پارامترهای $\boldsymbol{\theta}^{(k)}$ توسط معادله ۴-۲۳ ارائه شده است. این بردار، تغییرات تابع هزینه را نسبت به هر مؤلفه از $\boldsymbol{\theta}^{(k)}$ نشان می‌دهد و در الگوریتم‌های گرادیان نزولی برای به‌روزرسانی پارامترهای مدل استفاده می‌شود.

معادله ۴-۲۳. بردار گرادیان آنتروپی متقاطع برای کلاس k

$$\nabla_{\theta^{(k)}} J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \hat{p}_{k^{(i)}} - y_{k^{(i)}}\, x^{(i)}$$

© 2025 Mohsen Zare. All rights reserved.

اکنون می‌توانید بردار گرادیان را برای هر کلاس محاسبه کرده و سپس از گرادیان نزولی یا هر الگوریتم بهینه‌سازی دیگر استفاده کنید تا ماتریس پارامتر $\boldsymbol{\Theta}$ را پیدا کنید که تابع هزینه آنتروپی متقاطع را به حداقل می‌رساند.

بیایید رگرسیون سافت‌مکس را برای طبقه‌بندی گل‌های زنبق در هر سه کلاس استفاده کنیم. طبقه‌بند رگرسیون لجستیک در Scikit-Learn به‌طور خودکار رگرسیون سافت‌مکس را هنگام آموزش روی بیش از دو کلاس به کار می‌گیرد (به شرط استفاده از "solver="lbfgs که پیش‌فرض است). همچنین به‌طور پیش‌فرض از منظم‌سازی ℓ_2 بهره می‌برد که می‌توانید شدت آن را با هایپرپارامتر C کنترل کنید، همان‌طور که قبلاً توضیح داده شد:

```
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = iris["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
softmax_reg = LogisticRegression(C=30, random_state=42)
softmax_reg.fit(X_train, y_train)
```

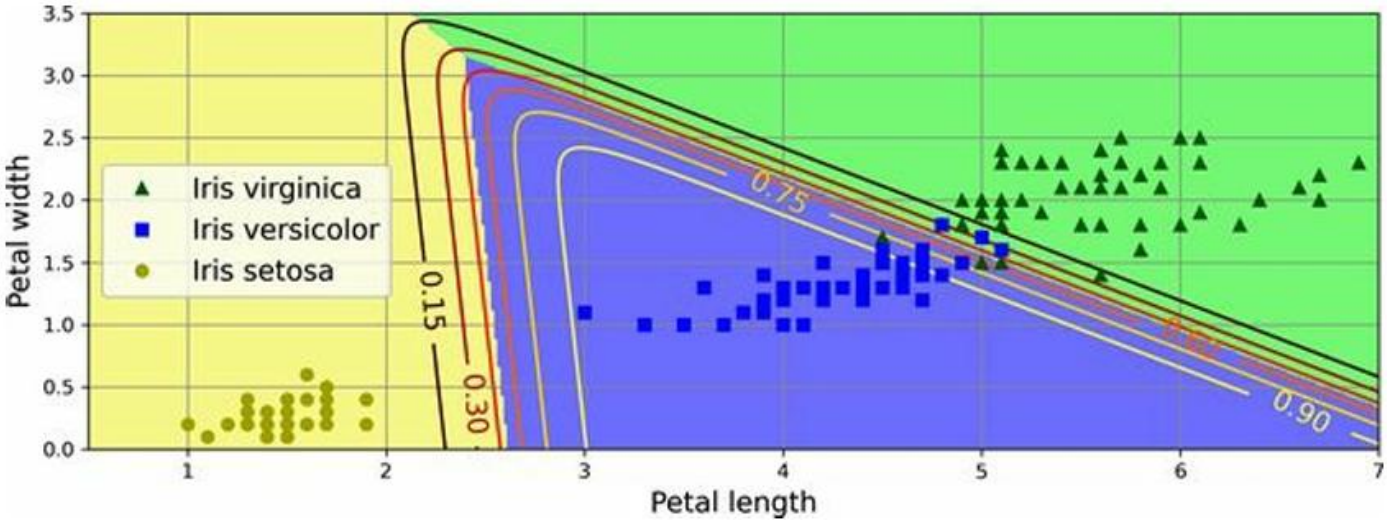
بنابراین، دفعه‌ی بعد که یک گل زنبق با طول گلبرگ ۵ سانتی‌متر و عرض ۲ سانتی‌متر دیدید، می‌توانید از مدل خود بخواهید نوع آن را پیش‌بینی کند. مدل با احتمال ۹۶% پیش‌بینی می‌کند که گل زنبق ویرجینیکا (کلاس ۲) است و با احتمال ۴% پیش‌بینی می‌کند که زنبق ورسیکالر باشد:

```
>>> softmax_reg.predict([[5, 2]])
array([2])
>>> softmax_reg.predict_proba([[5, 2]]).round(2)
array([[0. , 0.04, 0.96]])
```

شکل ۴-۲۵ مرزهای تصمیم‌گیری مدل را با رنگ‌های پس‌زمینه نشان می‌دهد. توجه داشته باشید که مرزهای بین هر دو کلاس به صورت خطی هستند. این شکل همچنین احتمالات کلاس *Iris versicolor* را با خطوط منحنی نمایش می‌دهد؛ برای مثال، خطی که با ۰٫۳۰ مشخص شده است، مرز احتمال ۳۰% را نشان می‌دهد. دقت کنید که مدل می‌تواند کلاسی را پیش‌بینی کند حتی اگر احتمال

© 2025 Mohsen Zare. All rights reserved.

تخمینی آن زیر ۵۰% باشد؛ برای مثال، در نقطه‌ای که همه مرزهای تصمیم‌گیری همگرا می‌شوند، احتمال تخمینی هر کلاس برابر ۳۳% است.



شکل ۴-۲۵. مرزهای تصمیم‌گیری رگرسیون سافت مکس

در این فصل، با روش‌های مختلف آموزش مدل‌های خطی برای رگرسیون و طبقه‌بندی آشنا شدید. شما یاد گرفتید چگونه از معادله فرم بسته برای حل رگرسیون خطی استفاده کنید و همچنین نحوه به‌کارگیری گرادیان نزولی را فرا گرفتید. علاوه بر این، دیدید که چگونه می‌توان انواع جریمه‌ها را به تابع هزینه اضافه کرد تا مدل منظم شود، منحنی‌های یادگیری را رسم و تحلیل کرد و روش توقف زودهنگام را پیاده‌سازی نمود. در نهایت، نحوه کار رگرسیون لجستیک و رگرسیون سافت‌مکس را آموختید. به این ترتیب، اولین جعبه سیاه یادگیری ماشین را باز کردیم! در فصل‌های بعدی، جعبه‌های سیاه بیشتری را بررسی خواهیم کرد که با ماشین‌های بردار پشتیبان شروع می‌شوند.

© 2025 Mohsen Zare. All rights reserved.

تمرین‌ها

۱. اگر یک مجموعه آموزشی با میلیون‌ها ویژگی داشته باشید، از کدام الگوریتم آموزشی رگرسیون خطی می‌توانید استفاده کنید؟

۲. فرض کنید ویژگی‌های موجود در مجموعه آموزشی شما مقیاس‌های بسیار متفاوتی دارند. کدام الگوریتم‌ها ممکن است از این مشکل رنج ببرند و چگونه؟ در مورد آن چه کاری می‌توانید انجام دهید؟

۳. آیا گرادیان نزولی می‌تواند هنگام آموزش یک مدل رگرسیون لجستیک در یک مینیمم محلی گیر کند؟

۴. آیا همه الگوریتم‌های گرادیان نزولی، در صورتی که به آنها اجازه دهید به مدت کافی اجرا شوند، به مدل یکسانی منجر می‌شوند؟

۵. فرض کنید از روش گرادیان نزولی دسته‌ای استفاده می‌کنید و خطای اعتبارسنجی را در هر دوره رسم می‌کنید. اگر متوجه شوید که خطای اعتبارسنجی به طور مداوم افزایش می‌یابد، احتمالاً چه اتفاقی افتاده است؟ چگونه می‌توانید این مشکل را برطرف کنید؟

۶. آیا ایده خوبی است که وقتی خطای اعتبارسنجی بالا می‌رود، فوراً روش گرادیان نزولی مینی-دسته‌ای را متوقف کنیم؟

۷. کدام الگوریتم گرادیان نزولی (از بین الگوریتم‌هایی که در موردشان صحبت کردیم) سریع‌تر به مجاورت راه‌حل بهینه می‌رسد؟ کدام یک واقعاً همگرا می‌شود؟ چگونه می‌توانید بقیه را نیز همگرا کنید؟

۸. فرض کنید از رگرسیون چندجمله‌ای استفاده می‌کنید. منحنی‌های یادگیری را رسم می‌کنید و متوجه می‌شوید که شکاف بزرگی بین خطای آموزش و خطای اعتبارسنجی وجود دارد. چه اتفاقی می‌افتد؟ سه راه برای حل این مشکل چیست؟

© 2025 Mohsen Zare. All rights reserved.

۹. فرض کنید از رگرسیون ریج استفاده می‌کنید و متوجه می‌شوید که خطای آموزش و خطای اعتبارسنجی تقریباً برابر و نسبتاً زیاد هستند. آیا می‌گویید که مدل از بایاس بالا یا واریانس بالا رنج می‌برد؟ آیا باید پارامتر منظم‌سازی α را افزایش دهید یا آن را کاهش دهید؟

۱۰. فرض کنید از رگرسیون چندجمله‌ای استفاده می‌کنید. منحنی‌های یادگیری را رسم می‌کنید و متوجه می‌شوید که شکاف بزرگی بین خطای آموزش و خطای اعتبارسنجی وجود دارد. چه اتفاقی می‌افتد؟ سه راه برای حل این مشکل چیست؟

۱۱. چرا از موارد زیر استفاده خواهید کرد:
الف. رگرسیون ریج به جای رگرسیون خطی ساده (یعنی بدون هیچ گونه منظم‌سازی)؟
ب. لاسو به جای رگرسیون ریج؟
ج. شبکه الاستیک به جای رگرسیون لاسو؟

۱۲. فرض کنید می‌خواهید تصاویر را به صورت فضای باز یا فضای داخلی و روز یا شب طبقه‌بندی کنید. آیا باید دو طبقه‌بندی‌کننده رگرسیون لجستیک یا یک طبقه‌بندی‌کننده رگرسیون سافت مکس پیاده‌سازی کنید؟

۱۳. پیاده‌سازی الگوریتم گرادیان نزولی دسته‌ای با الگوریتم توقف زودهنگام برای رگرسیون سافت-مکس بدون استفاده از Scikit-Learn و فقط با استفاده از NumPy. از آن در یک کار طبقه‌بندی مانند مجموعه داده iris استفاده کنید.

پاسخ‌های مربوط به این تمرین‌ها را می‌توانید در انتهای نوت‌بوک همین فصل، از طریق نشانی *https://homl.info/colab3* مشاهده کنید.

© 2025 Mohsen Zare. All rights reserved.

فصل ۵. ماشین‌های بردار پشتیبان

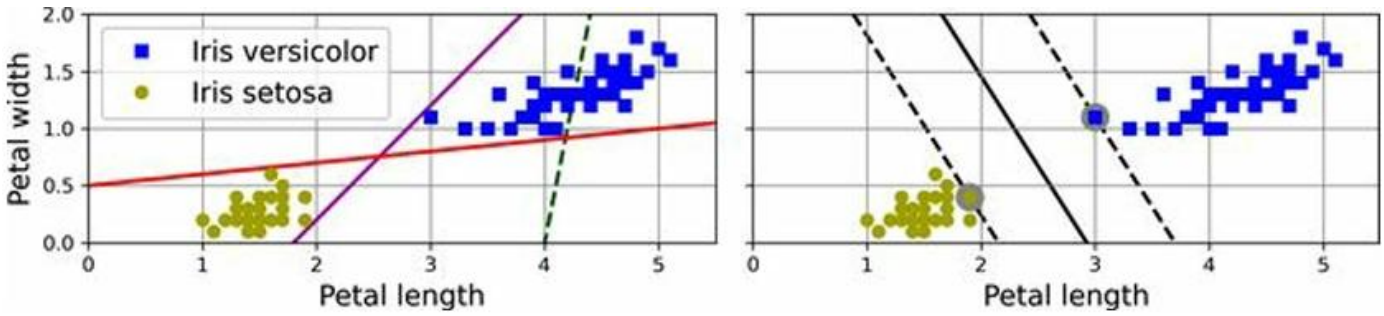
«ماشین بردار پشتیبان (SVM: *support vector machine*)» یک الگوریتم یادگیری ماشین قدرتمند و چندکاره است که می‌تواند برای طبقه‌بندی خطی و غیرخطی، رگرسیون و حتی شناسایی نمونه‌های جدید به‌کار گرفته شود. این مدل به‌ویژه در مسائل طبقه‌بندی با مجموعه داده‌های کوچک تا متوسط (یعنی چند صد تا چند هزار نمونه) عملکرد بسیار خوبی دارد. با این حال، همان‌طور که در ادامه خواهید دید، SVMها برای مجموعه داده‌های بسیار بزرگ چندان مقیاس‌پذیر نیستند.

در این فصل، با مفاهیم کلیدی ماشین‌های بردار پشتیبان، نحوه به‌کارگیری آن‌ها و مکانیزم عملکردشان آشنا می‌شویم. بیایید مستقیماً وارد مبحث شویم!

© 2025 Mohsen Zare. All rights reserved.

طبقه‌بندی خطی SVM

ایده‌ی بنیادی پشت ماشین‌های بردار پشتیبان (SVM) با تصاویر و نمودارها به‌خوبی قابل توضیح است. در شکل ۵-۱، بخشی از مجموعه داده‌های گل زنبق (معرفی‌شده در فصل ۴) نمایش داده شده است. در این مجموعه، دو کلاس به‌سادگی با یک خط مستقیم از هم جدا می‌شوند؛ یعنی داده‌ها «خطیِ قابل تفکیک (*linearly separable*)» و جداشدنی هستند–یعنی دو کلاس داده‌ها را می‌توان با یک خط (۲ بعد) یا ابرصفحه (n بعد) از هم جدا کرد. نمودار سمت چپ، مرزهای تصمیم‌گیری سه طبقه‌بند خطی ممکن را نشان می‌دهد. خط چین نشان‌دهنده مدلی است که ضعیف عمل می‌کند و حتی کلاس‌ها را درست از هم تفکیک نمی‌کند. دو مدل دیگر بر روی مجموعه آموزشی فعلی کاملاً عمل می‌کنند، اما مرز تصمیم‌گیری آن‌ها به نمونه‌ها بسیار نزدیک است و احتمال عملکرد ضعیف روی داده‌های جدید وجود دارد. در مقابل، خط ممتد در نمودار سمت راست مرز تصمیم‌گیری یک طبقه‌بندی‌کننده SVM را نمایش می‌دهد. این مرز نه تنها دو کلاس را جدا می‌کند، بلکه حداکثر فاصله را از نزدیک‌ترین نمونه‌های آموزشی حفظ می‌کند. می‌توان طبقه‌بندی‌کننده SVM را به‌عنوان ایجاد «عرضی‌ترین خیابان ممکن» بین کلاس‌ها تصور کرد، که با خط‌چین‌های موازی نشان داده شده است. این رویکرد را «طبقه‌بندی با حاشیه بزرگ (*large margin classification*)» می‌نامند–این روش باعث افزایش تعمیم مدل و کاهش احتمال اشتباه در داده‌های جدید می‌شود.



شکل ۵-۱. طبقه‌بندی با حاشیه بزرگ

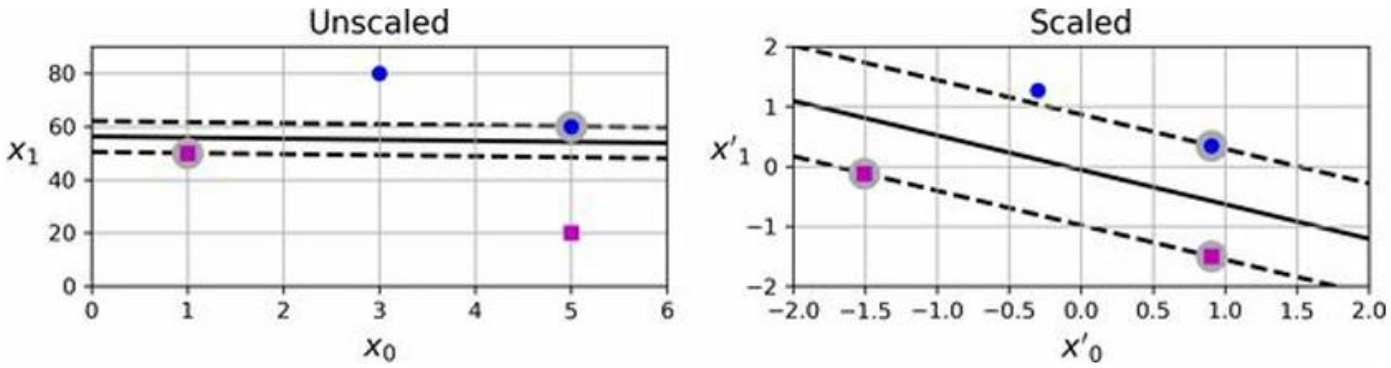
توجه داشته باشید که اضافه کردن نمونه‌های آموزشی بیشتر خارج از حاشیه‌ها (off the street) هیچ تأثیری بر مرز تصمیم‌گیری نخواهد داشت. در واقع، مرز تصمیم‌گیری تنها توسط نمونه‌هایی که درست بر لبه‌های حاشیه و روی مرز یا نزدیک به مرز تصمیم قرار دارند–یعنی «مماس با خطوط حاشیه

© 2025 Mohsen Zare. All rights reserved.

بالا یا پایین(edge of the street)»—تعیین یا «پشتیبانی» می‌شود. این نمونه‌ها به نام «بردارهای پشتیبان (*support vectors*)» شناخته می‌شوند و در شکل ۵-۱ با دایره مشخص شده‌اند.

هشدار

SVMها به مقیاس ویژگی‌ها بسیار حساس هستند، همان‌طور که در شکل ۲-۵ دیده می‌شود. در نمودار سمت چپ، مقیاس محور عمودی بسیار بزرگ‌تر از محور افقی است، بنابراین عریض‌ترین خیابان ممکن به سمت افقی کشیده شده است. پس از مقیاس‌بندی ویژگی‌ها—مثلاً با استفاده از StandardScaler در Scikit-Learn—مرز تصمیم‌گیری در نمودار سمت راست به شکل بهینه‌تری قرار می‌گیرد و عملکرد مدل بهبود می‌یابد.



شکل ۵-۲. حساسیت به مقیاس ویژگی‌ها

© 2025 Mohsen Zare. All rights reserved.

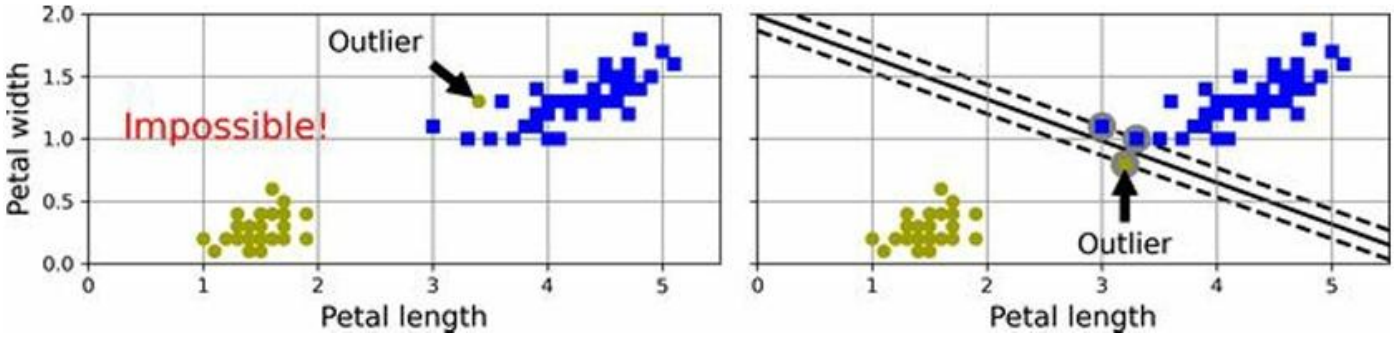
طبقه‌بندی حاشیه نرم

اگر به‌طور سختگیرانه‌ای فرض کنیم که همه نمونه‌ها باید کاملاً خارج از حاشیه‌ها و در سمت صحیح قرار بگیرند، به این حالت «طبقه‌بندی حاشیه سخت (*hard margin classification*)» گفته می‌شود. دو محدودیت اصلی این رویکرد عبارت‌اند از:

۱. تنها زمانی کاربرد دارد که داده‌ها کاملاً خطی قابل تفکیک باشند.

۲. بسیار به داده‌های پرت حساس است.

در شکل ۵-۳، مجموعه داده زنبق با یک نمونه پرت اضافه نشان داده شده است. در نمودار سمت چپ، ایجاد یک حاشیه سخت غیرممکن است؛ در نمودار سمت راست، مرز تصمیم‌گیری کاملاً متفاوت از شکل ۵-۱ بدون داده پرت است و مدل احتمالاً قابلیت تعمیم خوبی نخواهد داشت.



شکل ۵-۳. حساسیت حاشیه سخت به داده‌های پرت

برای رفع این محدودیت‌ها، از مدلی انعطاف‌پذیرتر استفاده می‌کنیم. ایده اصلی، یافتن تعادلی مناسب بین حداکثر کردن عرض حاشیه (خیابان) و محدود کردن «نقض حاشیه‌ای یا *margin violations*» (وقتی نمونه‌هایی داخل حاشیه یا حتی روی کلاس مخالف قرار گرفته‌اند و مرز تصمیم را نقض می‌کنند) است. این رویکرد به نام «طبقه‌بندی حاشیه نرم (*soft margin classification*)» شناخته می‌شود.

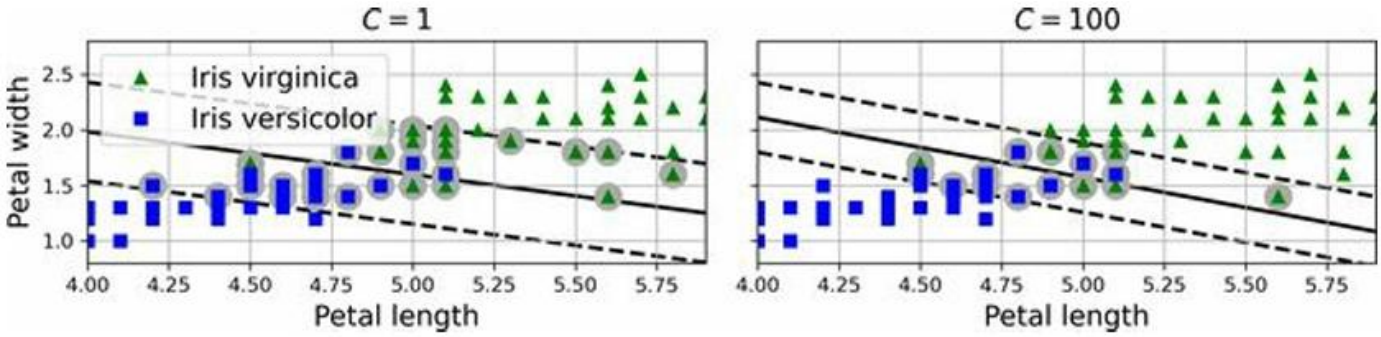
هنگام پیاده‌سازی یک SVM در Scikit-Learn، می‌توانید چندین هایپرپارامتر را تنظیم کنید، از جمله C:

© 2025 Mohsen Zare. All rights reserved.

- اگر C را کوچک انتخاب کنید، مدل مشابه سمت چپ شکل ۵-۴ خواهد بود: عرض حاشیه بزرگتر است، اما تعداد نقض حاشیه‌ای بیشتر می‌شود. این باعث می‌شود مدل نمونه‌های بیشتری را داخل خیابان بپذیرد و در نتیجه خطر بیش‌برازش کاهش یابد.

- اگر C بزرگ باشد، مدل مشابه سمت راست شکل ۵-۴ خواهد شد: مدل سعی می‌کند همه نمونه‌ها را دقیقاً در سمت صحیح حاشیه نگه دارد، که ممکن است به بیش‌برازش منجر شود.

به عبارت دیگر، کاهش C منجر به خیابان عریض‌تر و تخلفات بیشتر، و افزایش C منجر به خیابان تنگ‌تر و دقت بالاتر روی نمونه‌های آموزشی می‌شود. انتخاب مناسب C به تعادل بین پهنای حاشیه و تعمیم‌پذیری مدل بستگی دارد.



شکل ۵-۴. حاشیه بزرگ (چپ) در مقابل نقض حاشیه کمتر (راست)

نکته

اگر مدل SVM شما به بیش‌برازش تمایل دارد، می‌توانید C را کاهش دهید تا مدل منظم‌تر شود. کاهش C باعث می‌شود حاشیه گسترده‌تر شود و اجازه داده شود برخی نمونه‌ها داخل حاشیه یا حتی در سمت اشتباه قرار گیرند، که به بهبود تعمیم‌پذیری مدل کمک می‌کند.

کد زیر با استفاده از Scikit-Learn مجموعه داده زنبق (Iris) را بارگذاری کرده و یک طبقه‌بند SVM خطی برای تشخیص گل‌های *Iris virginica* آموزش می‌دهد. این pipeline ابتدا ویژگی‌ها را مقیاس‌بندی می‌کند و سپس یک LinearSVC با C=1 اعمال می‌کند:

© 2025 Mohsen Zare. All rights reserved.

```
from sklearn.datasets import load_iris
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = load_iris(as_frame=True)
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = (iris.target == 2) # Iris virginica

svm_clf = make_pipeline(StandardScaler(),
        LinearSVC(C=1, random_state=42))
svm_clf.fit(X, y)
```

مدل آموزش‌دیده، همان‌طور که در سمت چپ شکل ۵-۴ مشاهده می‌شود، آماده است. پس از آن، مانند هر مدل دیگر، می‌توانید از آن برای پیش‌بینی کلاس نمونه‌های جدید استفاده کنید:

```
>>> X_new = [[5.5, 1.7], [5.0, 1.5]]
>>> svm_clf.predict(X_new)
array([ True, False])
```

در این مثال، نمونه اول به‌عنوان *Iris virginica* طبقه‌بندی می‌شود، در حالی که نمونه دوم به این کلاس تعلق ندارد. برای درک نحوه تصمیم‌گیری مدل، می‌توانیم به امتیازاتی که SVM تولید کرده است نگاه کنیم. این امتیازات، فاصله علامت‌دار هر نمونه تا مرز تصمیم‌گیری را نشان می‌دهند؛ یعنی هرچه امتیاز مثبت‌تر و بزرگ‌تر باشد، نمونه با اطمینان بیشتری در کلاس مثبت قرار می‌گیرد و هرچه منفی‌تر باشد، نمونه به کلاس مخالف نزدیک‌تر است.

```
>>> svm_clf.decision_function(X_new)
array([ 0.66163411, -0.22036063])
```

برخلاف LogisticRegression، کلاس LinearSVC متد predict_proba() برای تخمین احتمالات کلاس ارائه نمی‌دهد. با این حال، اگر به جای آن از کلاس SVC استفاده کنید و هایپرپارامتر probability=True تنظیم شود، مدل پس از آموزش یک مدل اضافی برازش می‌دهد تا امتیازها تابع

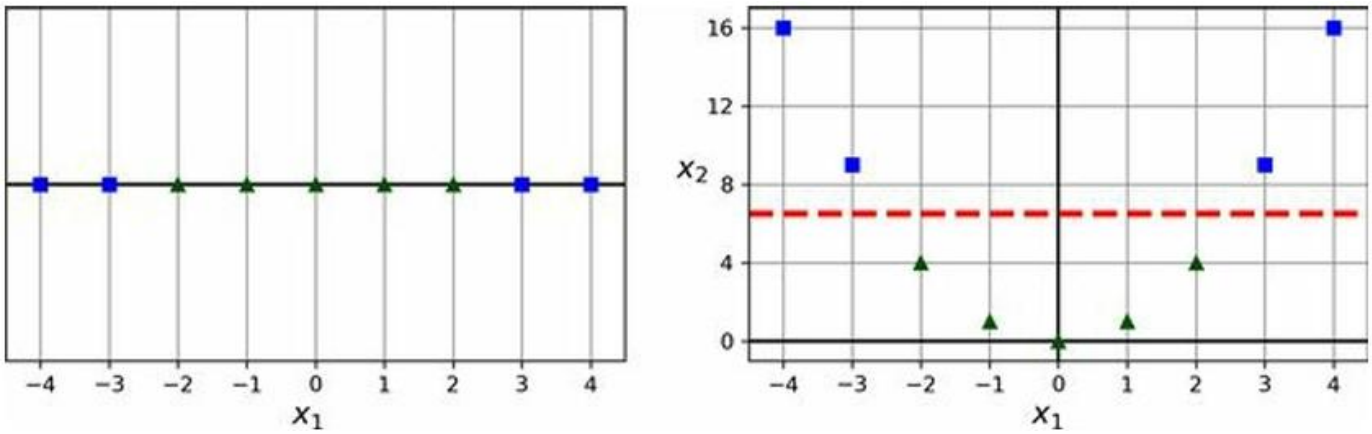
© 2025 Mohsen Zare. All rights reserved.

تصمیم SVM را به احتمالات تخمینی نگاشت(map) کند. در باطن، این فرآیند با اعتبارسنجی متقاطع ۵-بخشی (5-fold cross-validation) انجام می‌شود: ابتدا پیش‌بینی‌های خارج از نمونه (out-of-sample) برای هر نمونه در مجموعه آموزش تولید شده و سپس یک LogisticRegression روی این پیش‌بینی‌ها آموزش داده می‌شود. در نتیجه، آموزش کندتر خواهد بود، اما پس از آن، متدهای predict_proba() و predict_log_proba() قابل استفاده خواهند بود.

© 2025 Mohsen Zare. All rights reserved.

طبقه‌بندی SVM غیرخطی

اگرچه SVMهای خطی کارآمد هستند و اغلب عملکرد بسیار خوبی دارند، بسیاری از مجموعه داده‌ها به‌سادگی با یک خط قابل جداسازی نیستند. یکی از راهکارها برای مدیریت این نوع داده‌ها، افزودن ویژگی‌های جدید مانند ویژگی‌های چندجمله‌ای است (مشابه کاری که در فصل ۴ انجام دادیم). در برخی موارد، این کار باعث می‌شود داده‌ها به صورت خطی قابل تفکیک شوند. برای مثال، در نمودار سمت چپ شکل ۵-۵، یک مجموعه داده ساده با تنها یک ویژگی x_1 نشان داده شده است که به وضوح به صورت خطی قابل تفکیک نیست. اما اگر ویژگی دوم $x_2 = (x_1)^2$ اضافه شود، داده‌ها در فضای دوبعدی حاصل کاملاً قابل تفکیک به صورت خطی خواهند بود.



شکل ۵-۵. افزودن ویژگی‌ها برای جداسازی خطی یک مجموعه داده

برای پیاده‌سازی این ایده در Scikit-Learn، می‌توان یک pipeline ایجاد کرد که شامل مراحل زیر باشد:

۱. PolynomialFeatures برای اضافه کردن ویژگی‌های چندجمله‌ای، مشابه آنچه در رگرسیون چندجمله‌ای توضیح داده شد.

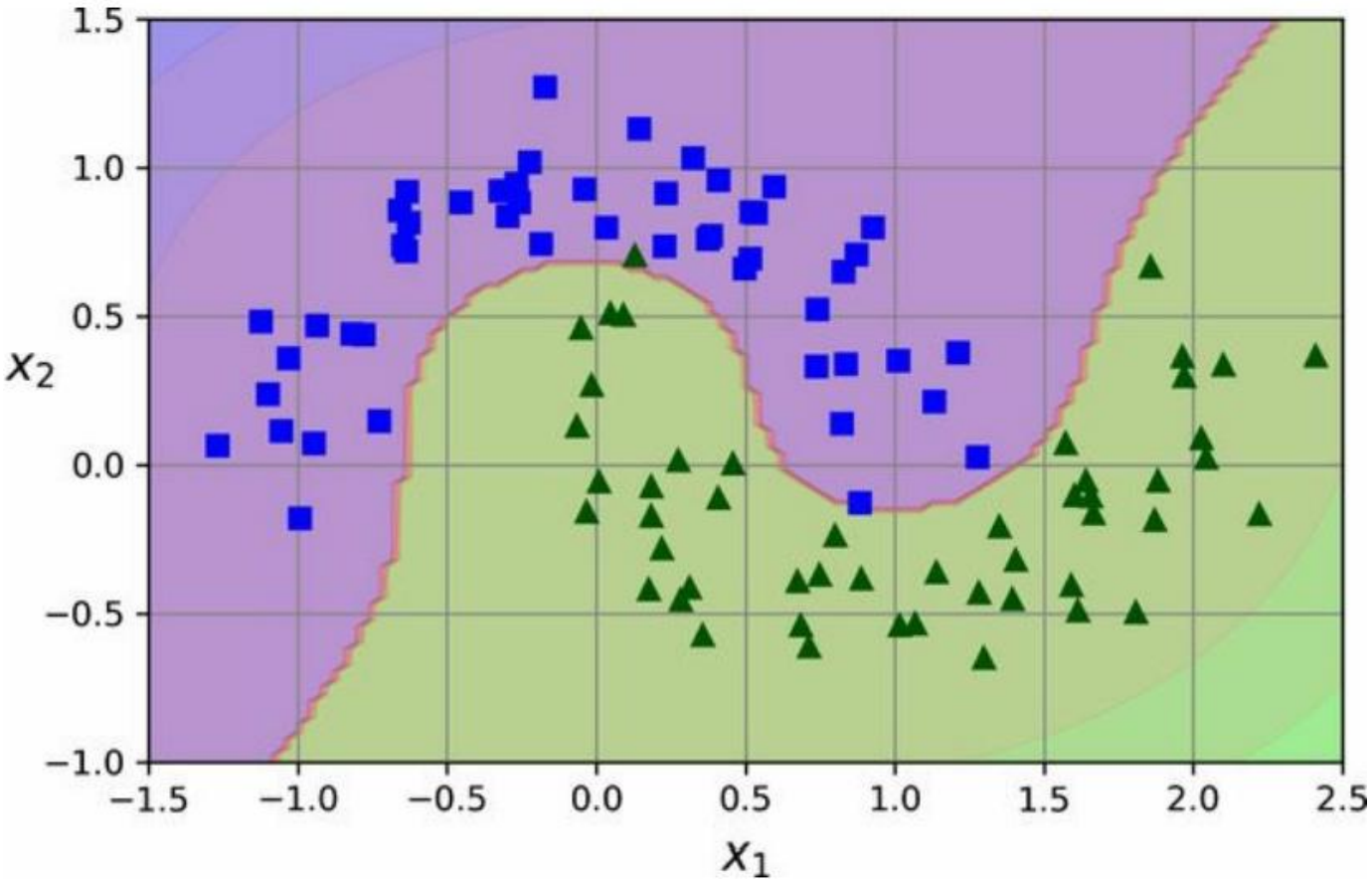
۲. StandardScaler برای مقیاس‌بندی ویژگی‌ها.

۳. LinearSVC به‌عنوان طبقه‌بند SVM خطی.

© 2025 Mohsen Zare. All rights reserved.

این روش را می‌توان روی مجموعه داده moons آزمایش کرد؛ مجموعه‌ای کوچک، ساده و آزمایشی برای طبقه‌بندی باینری که نقاط آن به شکل دو هلال ماه درهم‌تنیده قرار دارند. می‌توانید این مجموعه داده را با استفاده از تابع ()make_moons ایجاد کنید:

```
from sklearn.datasets import make_moons
from sklearn.preprocessing import PolynomialFeatures

X, y = make_moons(n_samples=100, noise=0.15, random_state=42)

polynomial_svm_clf = make_pipeline(
  PolynomialFeatures(degree=3),
  StandardScaler(),
  LinearSVC(C=10, max_iter=10_000, random_state=42))
polynomial_svm_clf.fit(X, y)
```



شکل ۵-۶. طبقه‌بند خطی SVM با استفاده از ویژگی‌های چندجمله‌ای

© 2025 Mohsen Zare. All rights reserved.

کرنل چندجمله‌ای

افزودن ویژگی‌های چندجمله‌ای روشی ساده برای پیاده‌سازی است و می‌تواند با انواع الگوریتم‌های یادگیری ماشین (نه صرفاً SVMها) عملکرد خوبی داشته باشد. با این حال، در درجه‌های پایین چندجمله‌ای، این روش توانایی مدیریت مجموعه داده‌های پیچیده را ندارد و در درجه‌های بالای چندجمله‌ای، تعداد ویژگی‌ها به شدت افزایش یافته و باعث کندی شدید مدل می‌شود.

خوشبختانه، در SVMها می‌توان از تکنیکی تقریبا جادویی به نام «ترفند کِرنِل (*kernel trick*)» استفاده کرد که بعداً در این فصل به تفصیل توضیح داده خواهد شد. ترفند کرنل اجازه می‌دهد بدون افزودن واقعی ویژگی‌های چندجمله‌ای—even با درجه بسیار بالا—به همان نتایجی برسیم که گویی این ویژگی‌ها اضافه شده‌اند؛ در واقع امکان جداسازی غیرخطی داده‌ها را می‌دهد و محاسبات پیچیده نگاشت مستقیم را حذف می‌کند(محاسبه فاصله/شباهت در فضای بالاتر بدون بردن مستقیم داده‌ها به آن فضا). بدین ترتیب، از انفجار ترکیبی تعداد ویژگی‌ها جلوگیری می‌شود. این قابلیت در کلاس SVC پیاده‌سازی شده و می‌توان آن را به‌راحتی روی مجموعه داده‌هایی مانند moons آزمایش کرد:

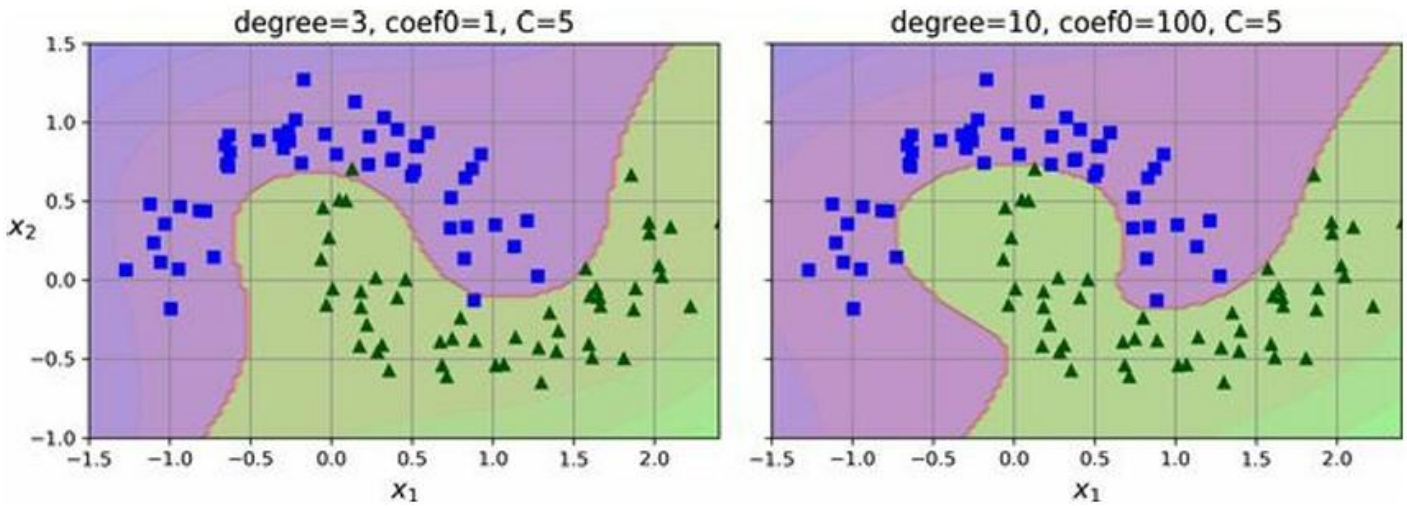
```
from sklearn.svm import SVC

poly_kernel_svm_clf = make_pipeline(StandardScaler(),
                    SVC(kernel="poly", degree=3, coef0=1, C=5))
poly_kernel_svm_clf.fit(X, y)
```

در این مثال، یک طبقه‌بندی‌کننده SVM با کرنل چندجمله‌ای درجه سوم آموزش داده شده است (سمت چپ شکل ۵-۷)، در حالی که سمت راست، یک SVM دیگر با کرنل چندجمله‌ای درجه دهم نمایش داده شده است. واضح است که اگر مدل دچار بیش‌برازش شود، کاهش درجه چندجمله‌ای مفید است و بالعکس، برای کم‌برازش می‌توان درجه را افزایش داد. همچنین، هایپرپارامتر coef0 تعیین می‌کند که مدل تا چه حد تحت تأثیر عبارات درجه بالا در مقابل عبارات درجه پایین قرار گیرد.

© 2025 Mohsen Zare. All rights reserved.

توجه

کرنل یک تابع است که شباهت یا inner product بین نمونه‌ها را محاسبه می‌کند.



شکل ۵-۷. طبقه‌بندهای SVM با هسته چندجمله‌ای

نکته

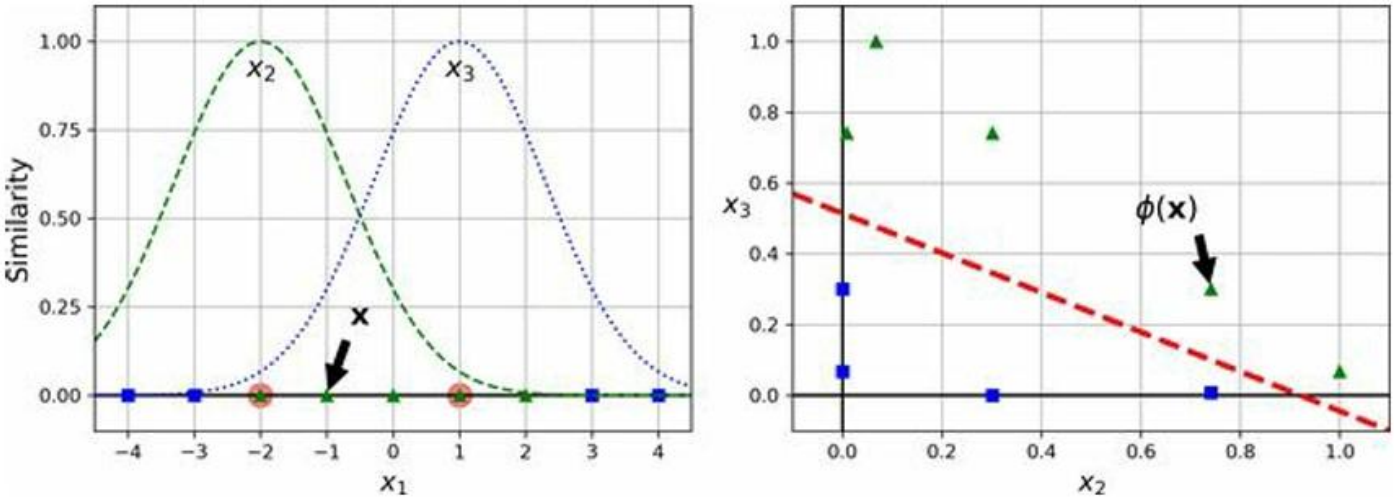
اگرچه بسیاری از هایپرپارامترها معمولاً به‌صورت خودکار (مثلاً با جستجوی تصادفی) تنظیم می‌شوند، آشنایی با عملکرد دقیق هر یک و نحوه تعامل آن‌ها با یکدیگر بسیار ارزشمند است؛ این دانش به شما امکان می‌دهد فضای جستجو را به شکل قابل‌توجهی محدود کرده و فرآیند بهینه‌سازی را مؤثرتر کنید.

© 2025 Mohsen Zare. All rights reserved.

ویژگی‌های مبتنی بر شباهت

راهکار دیگری برای مدیریت مسائل غیرخطی، افزودن ویژگی‌هایی است که میزان شباهت هر نمونه به « نقطه مرجع یا نقطه شاخص ($landmark$)» مشخص–نقاط کلیدی در منحنی یادگیری؛ نقاطی مشخص روی منحنی که تغییرات مهم عملکرد(مثل overfitting یا underfitting) را نشان می‌دهند.–را اندازه‌گیری می‌کنند؛ مشابه روشی که در فصل ۲ برای ویژگی‌های شباهت جغرافیایی استفاده کردیم. برای نمونه، مجموعه داده یک‌بعدی قبلی را در نظر بگیرید و دو نقطه عطف در $x_1 = -2$ و $x_1 = 1$ اضافه کنید (نمودار سمت چپ شکل ۵-۸). سپس می‌توانیم یک تابع شباهت بر اساس پایه شعاعی گاوسی (Gaussian RBF) با $\gamma = 0.3$ تعریف کنیم. این تابع به شکل زنگوله است و مقدار آن بین ۰ (برای نمونه‌های بسیار دور از نقطه عطف) تا ۱ (برای نمونه دقیقاً روی نقطه عطف) تغییر می‌کند.

اکنون می‌توانیم ویژگی‌های جدید را محاسبه کنیم. به‌عنوان مثال، نمونه $x_1 = -1$ را در نظر بگیرید: این نمونه در فاصله ۱ واحدی از اولین نقطه عطف و ۲ واحدی از دومین نقطه عطف قرار دارد. بنابراین، ویژگی‌های جدید آن به صورت $x_2 = exp(-0.3 \times 1^2) \approx 0.74$ و $x_3 = exp(-0.3 \times 2^2) \approx 0.30$ خواهند بود. نمودار سمت راست شکل ۵-۸ مجموعه داده پس از تبدیل (با حذف ویژگی‌های اصلی) را نشان می‌دهد؛ همان‌طور که مشاهده می‌کنید، اکنون داده‌ها به شکل خطی قابل تفکیک هستند.



شکل ۵-۸. ویژگی‌های شباهت با استفاده از RBF گاوسی

© 2025 Mohsen Zare. All rights reserved.

ممکن است این سؤال پیش بیاید که نقاط عطف را چگونه باید انتخاب کرد. ساده‌ترین روش، ایجاد یک نقطه عطف در موقعیت هر نمونه در مجموعه داده است. این کار باعث افزایش تعداد ابعاد می‌شود و احتمال خطی قابل تفکیک شدن مجموعه آموزشی تبدیل‌شده را افزایش می‌دهد. با این حال، عیب این روش این است که اگر مجموعه‌ای شامل m نمونه و n ویژگی داشته باشید، پس از تبدیل (با حذف ویژگی‌های اصلی) مجموعه به m نمونه و m ویژگی خواهد رسید. بنابراین، اگر مجموعه آموزشی بسیار بزرگ باشد، تعداد ویژگی‌ها نیز به همان میزان افزایش خواهد یافت.

© 2025 Mohsen Zare. All rights reserved.

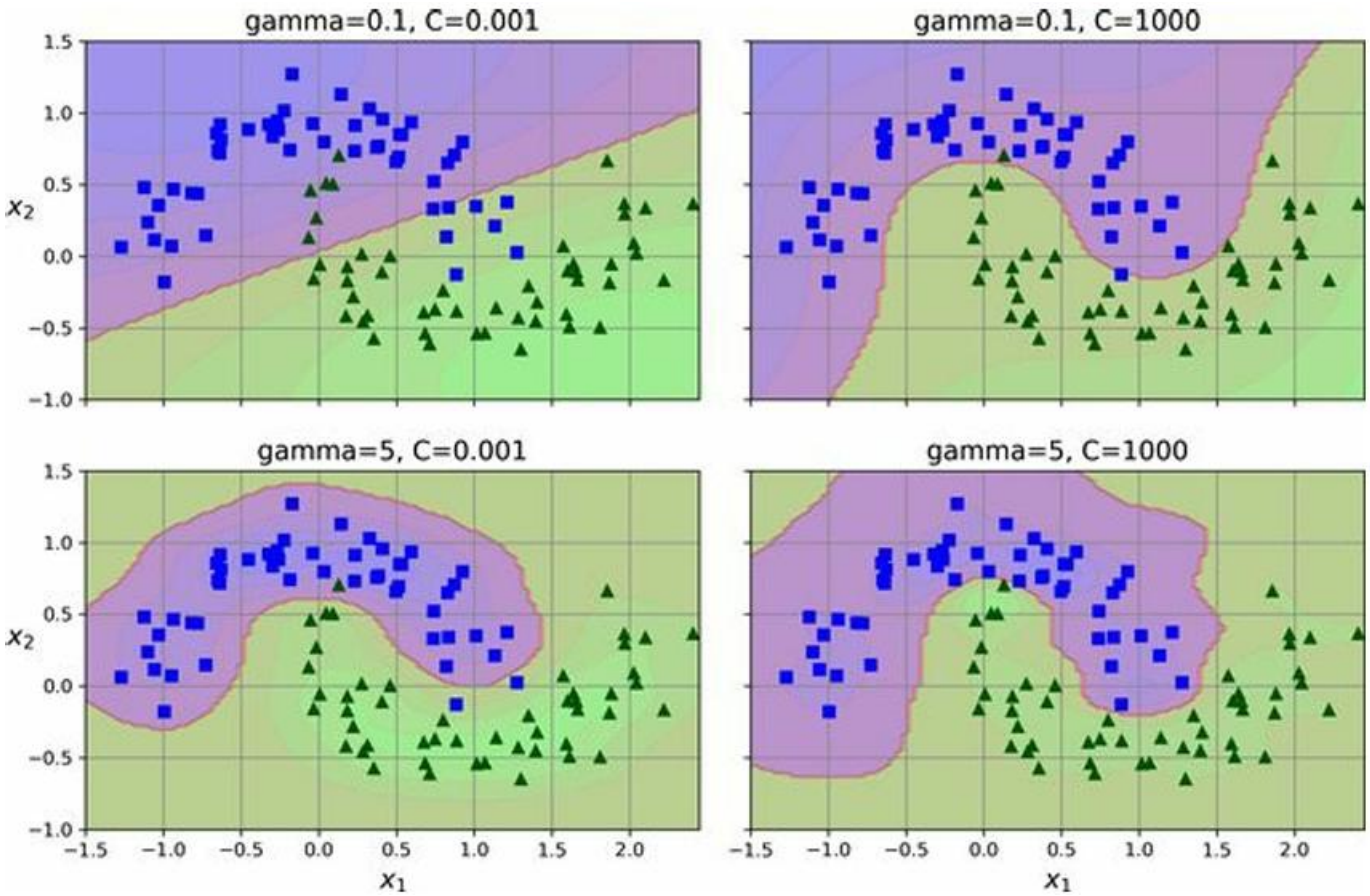
کرنل RBF گاوسی

مشابه رویکرد ویژگی‌های چندجمله‌ای، استفاده از ویژگی‌های مبتنی بر شباهت می‌تواند برای هر الگوریتم یادگیری ماشین مفید باشد، اما محاسبه همه این ویژگی‌های اضافی—به‌ویژه در مجموعه‌های آموزشی بزرگ—ممکن است بسیار پرهزینه باشد. در اینجا نیز «ترفند کرنل» جادوی SVM را به کار می‌گیرد و امکان دستیابی به همان نتیجه را فراهم می‌کند، انگار که تعداد زیادی ویژگی شباهت اضافه کرده‌اید، بدون اینکه واقعاً این کار انجام شده باشد. برای نمونه، می‌توانیم از کلاس SVC با کرنل Gaussian RBF استفاده کنیم.

```
rbf_kernel_svm_clf = make_pipeline(StandardScaler(),
                    SVC(kernel="rbf", gamma=5, C=0.001))
rbf_kernel_svm_clf.fit(X, y)
```

این مدل در پایین سمت چپ شکل ۵-۹ نشان داده شده است. نمودارهای دیگر، مدل‌های آموزش‌دیده با مقادیر مختلف هایپرپارامترهای γ و C را نمایش می‌دهند. افزایش γ، منحنی زنگوله‌ای شکل تابع RBF را باریک‌تر می‌کند (همان‌طور که در شکل ۵-۸ دیده شد)، در نتیجه تأثیر هر نمونه محدودتر شده و مرز تصمیم‌گیری پیچیده‌تر و به‌طور محلی حول نمونه‌ها می‌چرخد. برعکس، مقدار γ کوچک، منحنی زنگوله‌ای را گسترده‌تر می‌کند، بنابراین نمونه‌ها دامنه تأثیر بیشتری دارند و مرز تصمیم‌گیری هموارتر خواهد بود. از این رو، γ مانند یک هایپرپارامتر تنظیم‌کننده عمل می‌کند: اگر مدل بیش‌برازش دارد، γ را کاهش دهید و اگر کم‌برازش است، γ را افزایش دهید—مشابه نقش هایپرپارامتر C.

© 2025 Mohsen Zare. All rights reserved.



شکل ۵-۹. طبقه‌بندهای SVM با استفاده از هسته RBF

علاوه بر هسته‌های متداول، هسته‌های دیگری نیز وجود دارند که کمتر کاربرد دارند و اغلب برای ساختارهای داده خاص طراحی شده‌اند. برای مثال، «هسته‌های رشته‌ای» (*String kernels*) گاهی در طبقه‌بندی اسناد متنی یا توالی‌های DNA به کار می‌روند، مانند هسته زیرتوالی رشته‌ای یا هسته‌های مبتنی بر فاصله لونشتاین (Levenshtein).

© 2025 Mohsen Zare. All rights reserved.

نکته

با این همه گزینه برای انتخاب کرنل، چگونه تصمیم بگیریم؟ به طور کلی، همیشه ابتدا باید کرنل خطی را امتحان کنید. کلاس LinearSVC بسیار سریع‌تر از SVC(kernel="linear") عمل می‌کند، به‌ویژه برای مجموعه داده‌های بزرگ. اگر مجموعه داده نسبتاً کوچک یا متوسط است، می‌توانید SVMهای هسته‌دار را نیز امتحان کنید و معمولاً با هسته Gaussian RBF شروع کنید، چرا که اغلب نتایج خوبی می‌دهد. سپس، در صورت وجود زمان و منابع محاسباتی، می‌توانید با جستجوی هایپرپارامترها، هسته‌های دیگر را نیز آزمایش کنید. همچنین اگر کرنل‌هایی طراحی‌شده مخصوص ساختار داده مجموعه آموزشی شما وجود دارند، حتماً آن‌ها را هم بررسی کنید.

© 2025 Mohsen Zare. All rights reserved.

کلاس‌های SVM و پیچیدگی محاسباتی

کلاس LinearSVC مبتنی بر کتابخانه liblinear است که الگوریتمی بهینه‌شده برای SVMهای خطی ارائه می‌دهد. این کلاس از ترفند کرنل پشتیبانی نمی‌کند، اما تقریباً به‌طور خطی با تعداد نمونه‌ها و ویژگی‌ها مقیاس می‌یابد. پیچیدگی زمانی آموزش آن تقریباً $O(m \times n)$ است. اگر دقت بسیار بالایی مدنظر باشد، زمان آموزش افزایش می‌یابد که این موضوع توسط هایپرپارامتر تُلِرانس ϵ (در Scikit-Learn با نام tol) کنترل می‌شود. در بیشتر مسائل طبقه‌بندی، مقدار پیش‌فرض تلرانس به‌خوبی عمل می‌کند.

کلاس SVC مبتنی بر کتابخانه libsvm است و الگوریتمی که از ترفند کرنل پشتیبانی می‌کند را ارائه می‌دهد. پیچیدگی زمانی آموزش آن معمولاً بین $O(m^2 \times n)$ و $O(m^3 \times n)$ قرار دارد. بنابراین، وقتی تعداد نمونه‌های آموزشی زیاد باشد (مثلاً صدها نمونه)، آموزش به‌شدت کند می‌شود و این الگوریتم برای مجموعه‌های آموزشی غیرخطی کوچک تا متوسط مناسب‌ترین گزینه است. از سوی دیگر، با افزایش تعداد ویژگی‌ها، به‌ویژه در ویژگی‌های پراکنده (یعنی وقتی هر نمونه تعداد کمی ویژگی غیرصفر دارد)، الگوریتم به خوبی مقیاس‌پذیر است و تقریباً با میانگین تعداد ویژگی‌های غیرصفر در هر نمونه همگام می‌شود.

کلاس SGDClassifier به‌صورت پیش‌فرض «طبقه‌بندی با حاشیه بزرگ» انجام می‌دهد و امکان تنظیم هایپرپارامترهای آن—به‌ویژه هایپرپارامترهای منظم‌سازی (مانند آلفا و نوع جریمه) و نرخ یادگیری—وجود دارد تا نتایجی مشابه SVMهای خطی ارائه دهد. این کلاس برای آموزش از گرادیان نزولی تصادفی (SGD) استفاده می‌کند (به فصل ۴ مراجعه کنید)، که یادگیری افزایشی و مصرف کم حافظه را ممکن می‌سازد؛ بنابراین می‌توان از آن برای مجموعه داده‌های بزرگ که در حافظه RAM جا نمی‌شوند (یادگیری خارج از هسته–یادگیری خارج از حافظه اصلی؛ مناسب برای داده‌های حجیم و پردازش تدریجی؛ الگوریتم‌ها داده‌ها را بصورت دسته‌ای (batch) یا جریان داده (streaming) پردازش می‌کنند تا بتوانند روی داده‌های بزرگ‌تر از حافظه RAM آموزش ببینند) استفاده کرد. علاوه بر این، SGDClassifier به خوبی مقیاس‌پذیر است، زیرا پیچیدگی محاسباتی آن $O(m \times n)$ است. جدول ۱-۵ مقایسه‌ای از کلاس‌های مختلف طبقه‌بندی SVM در Scikit-Learn ارائه می‌دهد.

© 2025 Mohsen Zare. All rights reserved.

جدول ۵-۱. مقایسه کلاس‌های Scikit-Learn برای طبقه‌بندی SVM

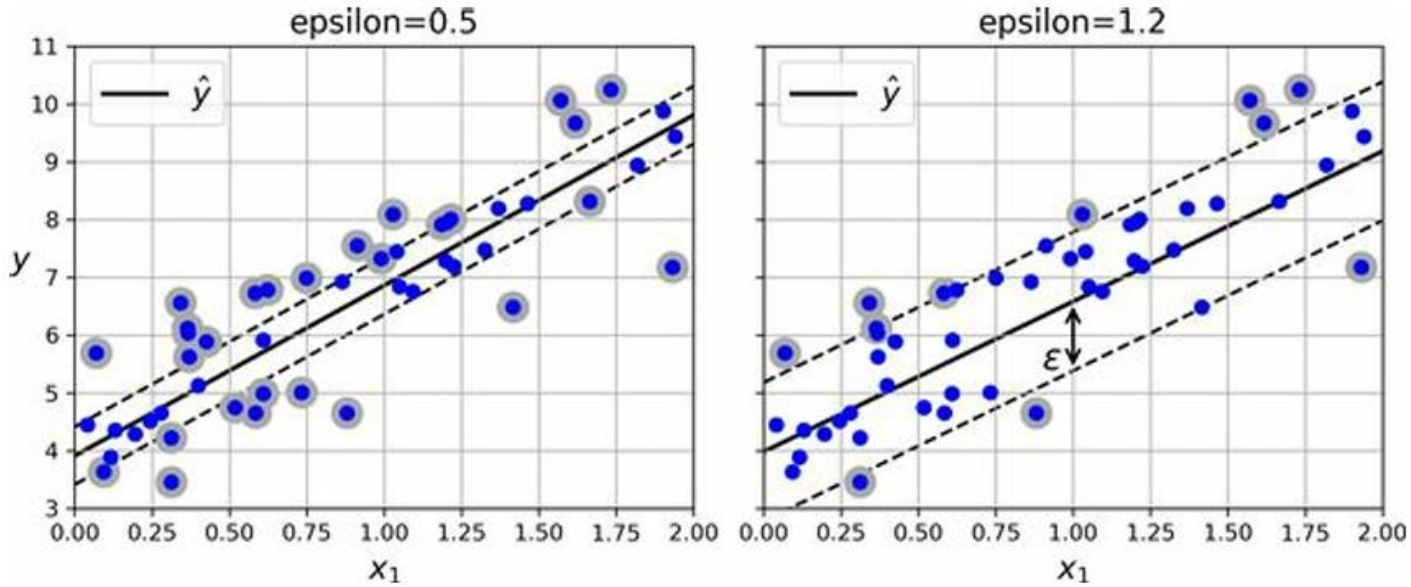
کلاس	پیچیدگی زمانی	پشتیبانیِ خارج از حافظه اصلی	نیاز به مقیاس‌بندی	ترفند کرنل
LinearSVC	$O(m \times n)$	خیر	بله	خیر
SVC	$O(m^2 \times n)\ to\ O(m^3 \times n)$	خیر	بله	بله
SGDClassifier	$O(m \times n)$	بله	بله	خیر

حال بیایید ببینیم که چگونه می‌توان از الگوریتم‌های SVM برای رگرسیون خطی و غیرخطی نیز استفاده کرد.

© 2025 Mohsen Zare. All rights reserved.

رگرسیون SVM

برای استفاده از SVM در مسائل رگرسیون به جای طبقه‌بندی، رویکرد کمی تغییر می‌کند: به جای یافتن بزرگ‌ترین خیابان ممکن بین دو کلاس و محدود کردن «نقض حاشیه‌ای(*margin violations*)»، رگرسیون SVM تلاش می‌کند تا بیشترین تعداد نمونه‌ها را «داخل» خیابان(on the street) نگه دارد و همزمان نقض حاشیه‌ای (نمونه‌هایی که «خارج» از خیابان(off the street) قرار می‌گیرند) را به حداقل برساند. عرض خیابان توسط یک هایپرپارامتر ϵ کنترل می‌شود. شکل ۵-۱۰ دو مدل رگرسیون SVM خطی را نمایش می‌دهد که بر روی داده‌های خطی آموزش دیده‌اند: یکی با حاشیه کوچک ($\epsilon = 0.5$) و دیگری با حاشیه بزرگ‌تر ($\epsilon = 1.2$). مدل با حاشیه کوچک سعی می‌کند نمونه‌ها را دقیق‌تر برازش دهد، در حالی که مدل با حاشیه بزرگ‌تر انعطاف‌پذیری بیشتری دارد و نوسانات کمتری را برازش می‌کند.



شکل ۵-۱۰. رگرسیون SVM

کاهش مقدار ϵ باعث افزایش تعداد بردارهای پشتیبان می‌شود و در نتیجه مدل منظم‌تر می‌شود. علاوه بر این، افزودن نمونه‌های آموزشی بیشتر در محدوده حاشیه هیچ تأثیری بر پیش‌بینی‌های مدل ندارد؛ به همین دلیل، این نوع رگرسیون را ϵ-غیرِحساس (*ϵ-insensitive*) می‌نامند(مدل فقط خطاهای

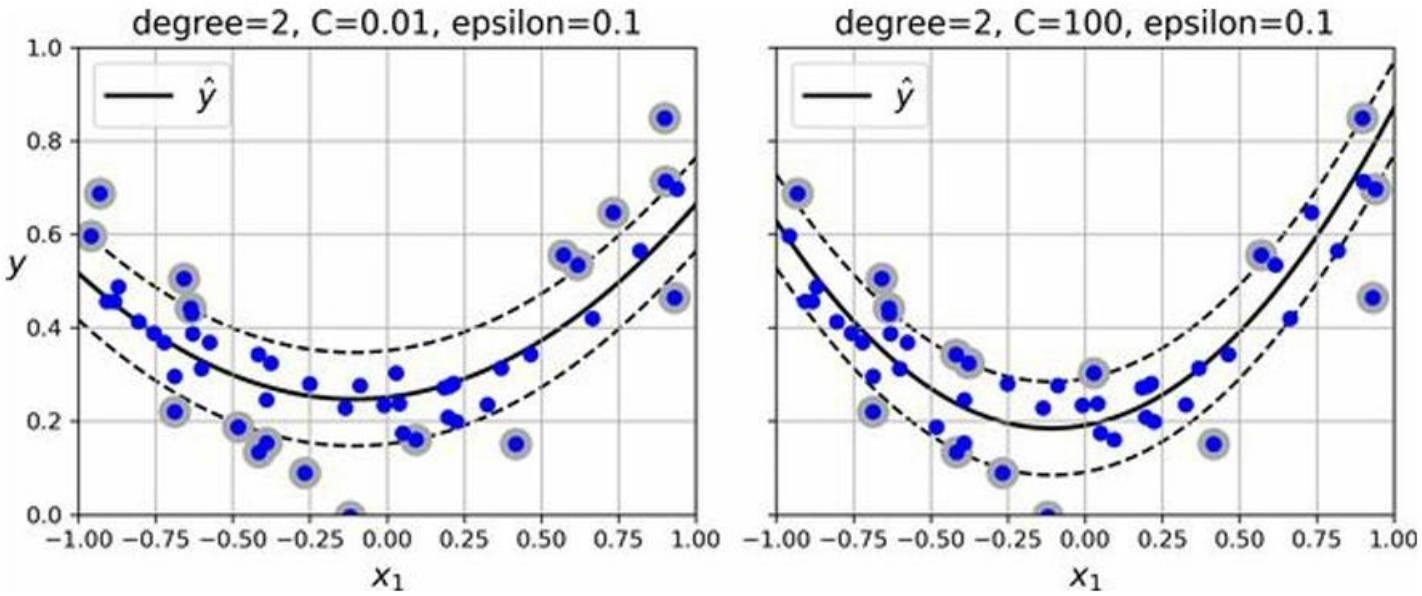
© 2025 Mohsen Zare. All rights reserved.

بزرگ‌تر از ϵ را جریمه می‌کند؛ بی‌تفاوت بودن نسبت به خطاهای کوچک؛ این باعث می‌شود مدل ساده‌تر شود و نویز کم‌اهمیت را نادیده بگیرد).

برای پیاده‌سازی رگرسیون خطی SVM می‌توانید از کلاس LinearSVR در Scikit-Learn استفاده کنید. این کلاس امکان آموزش مدل‌هایی مانند نمونه‌ی سمت چپ شکل ۵-۱۰ را فراهم می‌کند:

```
from sklearn.svm import LinearSVR

X, y = [...] # a linear dataset
svm_reg = make_pipeline(StandardScaler(),
        LinearSVR(epsilon=0.5, random_state=42))
svm_reg.fit(X, y)
```

برای انجام رگرسیون غیرخطی می‌توان از مدل SVM کرنل‌سازی‌شده استفاده کرد(نسخه پیشرفته SVM که با استفاده از kernel trick امکان جداسازی داده‌های غیرخطی در فضاهای با بُعد بالا را فراهم می‌کند.). شکل ۵-۱۱ نمونه‌ای از رگرسیون SVM روی یک مجموعه داده درجه دوم تصادفی را نشان می‌دهد که از هسته چندجمله‌ای درجه دوم بهره می‌برد. در نمودار سمت چپ، مدل دارای میزان منظم‌سازی متوسط (C کوچک) است، در حالی که در نمودار سمت راست، منظم‌سازی کمتر و مقدار C بزرگ‌تر اعمال شده است.



شکل ۵-۱۱. رگرسیون SVM با استفاده از کرنل چندجمله‌ای درجه دو

© 2025 Mohsen Zare. All rights reserved.

برای تولید مدلی مشابه نمودار سمت چپ شکل ۵-۱۱، می‌توان از کلاس SVR در Scikit-Learn استفاده کرد که از ترفند کرنل پشتیبانی می‌کند:

```
from sklearn.svm import SVR

X, y = [...] # a quadratic dataset
svm_poly_reg = make_pipeline(StandardScaler(),
            SVR(kernel="poly", degree=2, C=0.01, epsilon=0.1))
svm_poly_reg.fit(X, y)
```

کلاس SVR همانند کلاس SVC برای رگرسیون کرنل‌دار عمل می‌کند، در حالی که LinearSVR معادل خطی آن است و مشابه LinearSVC رفتار می‌کند. به طور خاص، LinearSVR به شکل خطی با اندازه مجموعه آموزشی مقیاس‌پذیر است، اما SVR همانند SVC هنگام افزایش تعداد نمونه‌ها به شدت کند می‌شود.

توجه

همان‌طور که در فصل ۹ بررسی خواهیم کرد، SVMها همچنین قابلیت استفاده برای تشخیص نقاط یا نمونه‌های جدید را نیز دارند.

© 2025 Mohsen Zare. All rights reserved.

تمرین ها

۱. ایده اساسی پشت ماشین‌های بردار پشتیبان چیست؟

۲. بردار پشتیبان چیست؟

۳. چرا هنگام استفاده از SVM ها، مقیاس‌بندی ورودی‌ها مهم است؟

۴. آیا یک طبقه‌بندی SVM می‌تواند هنگام طبقه‌بندی یک نمونه، امتیاز اطمینان ارائه دهد؟ در مورد احتمال چطور؟

۵. چگونه می‌توانید بین LinearSVC، SVC و SGDClassifier یکی را انتخاب کنید؟

۶. فرض کنید یک طبقه‌بند SVM را با کرنل RBF آموزش داده‌اید، اما به نظر می‌رسد که با مجموعه آموزشی تطابق کمتری دارد. آیا باید γ (گاما) را افزایش یا کاهش دهید؟ در مورد C چطور؟

۷. منظور از غیرحساس بودن یک مدل به ϵ (ϵ-*insensitive*) چیست؟

۸. فایده استفاده از ترفند کرنل چیست؟

۹. یک LinearSVC را روی یک مجموعه داده‌ی تفکیک‌پذیرِ خطی آموزش دهید. سپس یک SVC و یک SGDClassifier را روی همان مجموعه داده آموزش دهید. ببینید آیا می‌توانید کاری کنید که آنها تقریباً مدل یکسانی تولید کنند یا خیر.

۱۰. یک طبقه‌بند SVM را روی مجموعه داده wine آموزش دهید، که می‌توانید آن را با استفاده از sklearn.datasets.load_wine() بارگذاری کنید. این مجموعه داده شامل آنالیزهای شیمیایی ۱۷۸ نمونه شراب تولید شده توسط ۳ تولیدکننده مختلف است: هدف، آموزش یک مدل طبقه‌بندی است که بتواند تولیدکننده را بر اساس آنالیز شیمیایی شراب پیش‌بینی کند. از آنجایی که طبقه‌بندهای SVM طبقه‌بندهای باینری هستند، برای طبقه‌بندی هر سه کلاس باید از روش یک در مقابل همه استفاده کنید. به چه دقتی می‌توانید برسید؟

۱۱. یک رگرسیونر SVM را روی مجموعه داده‌های مسکن کالیفرنیا آموزش داده و تنظیم دقیق(fine-tune) کنید. شما می‌توانید از مجموعه داده اصلی به جای نسخه اصلاح

© 2025 Mohsen Zare. All rights reserved.

شده‌ای که در فصل ۲ استفاده کردیم، استفاده کنید که می‌توانید آن را با استفاده از sklearn.datasets.fetch_california_housing() بارگذاری کنید. اهداف، صدها هزار دلار را نشان می‌دهند. از آنجایی که بیش از ۲۰،۰۰۰ نمونه وجود دارد، SVM ها می‌توانند کند باشند، بنابراین برای تنظیم هایپرپارامتر باید از نمونه‌های بسیار کمتری (مثلاً ۲۰۰۰) استفاده کنید تا ترکیبات هایپرپارامتر بیشتری را آزمایش کنید. RMSE بهترین مدل شما چیست؟

پاسخ‌های مربوط به این تمرین‌ها را می‌توانید در انتهای نوت‌بوک همین فصل، از طریق نشانی *https://homl.info/colab3* مشاهده کنید.

© 2025 Mohsen Zare. All rights reserved.

فصل ۶. درخت‌های تصمیم‌گیری

«درخت‌های تصمیم (*Decision Trees*)» از جمله الگوریتم‌های همه‌کاره یادگیری ماشین هستند که می‌توانند برای طبقه‌بندی، رگرسیون و حتی مسائل چندخروجی به کار روند. این مدل‌ها توانمند هستند و مجموعه داده‌های پیچیده را می‌توانند به خوبی برازش دهند. برای نمونه، در فصل ۲ یک مدل DecisionTreeRegressor روی مجموعه داده‌های مسکن کالیفرنیا آموزش داده شد و به حدی دقیق برازش شد که در واقع بیش‌برازش رخ داد.

علاوه بر این، درخت‌های تصمیم، بلوک‌های اصلی جنگل‌های تصادفی (Random Forests) هستند که در فصل ۷ مورد بحث قرار خواهند گرفت و از قدرتمندترین الگوریتم‌های یادگیری ماشین به شمار می‌روند.

در این فصل، ابتدا به آموزش، مصورسازی و پیش‌بینی با درخت‌های تصمیم می‌پردازیم. سپس الگوریتم CART که در Scikit-Learn استفاده می‌شود را بررسی می‌کنیم، و نحوه منظم‌سازی درخت‌ها و استفاده از آن‌ها برای رگرسیون را توضیح خواهیم داد. در نهایت، برخی از محدودیت‌های درخت‌های تصمیم‌گیری را مرور خواهیم کرد.

© 2025 Mohsen Zare. All rights reserved.

آموزش و مصورسازی درخت تصمیم

برای فهم بهتر درخت‌های تصمیم، ابتدا یک درخت می‌سازیم و نحوه پیش‌بینی آن را بررسی می‌کنیم. برای مثال، می‌توان یک DecisionTreeClassifier را روی مجموعه داده Iris آموزش داد (که در فصل ۴ معرفی شد). این روند به شما اجازه می‌دهد تا ساختار درخت، گره‌ها و نحوه تصمیم‌گیری مدل بر اساس ویژگی‌های گل‌ها را مشاهده کنید و درک عمیق‌تری از مکانیزم پیش‌بینی درخت‌های تصمیم پیدا کنید:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris(as_frame=True)
X_iris = iris.data[["petal length (cm)", "petal width (cm)"]].values
y_iris = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_iris, y_iris)
```

پس از آموزش درخت تصمیم، می‌توان آن را به تصویر کشید. یک روش رایج استفاده از تابع export_graphviz() است که ساختار درخت را به یک فایل گراف با فرمت dot. صادر می‌کند، برای مثال با نام *iris_tree.dot* . این فایل گراف حاوی اطلاعات کامل در مورد گره‌ها، شاخه‌ها و معیارهای تصمیم‌گیری درخت است و می‌توان آن را با ابزارهای مناسب به نمودار قابل مشاهده تبدیل کرد:

```
from sklearn.tree import export_graphviz

export_graphviz(
    tree_clf,
    out_file="iris_tree.dot",
    feature_names=["petal length (cm)", "petal width (cm)"],
    class_names=iris.target_names,
    rounded=True,
    filled=True
  )
```

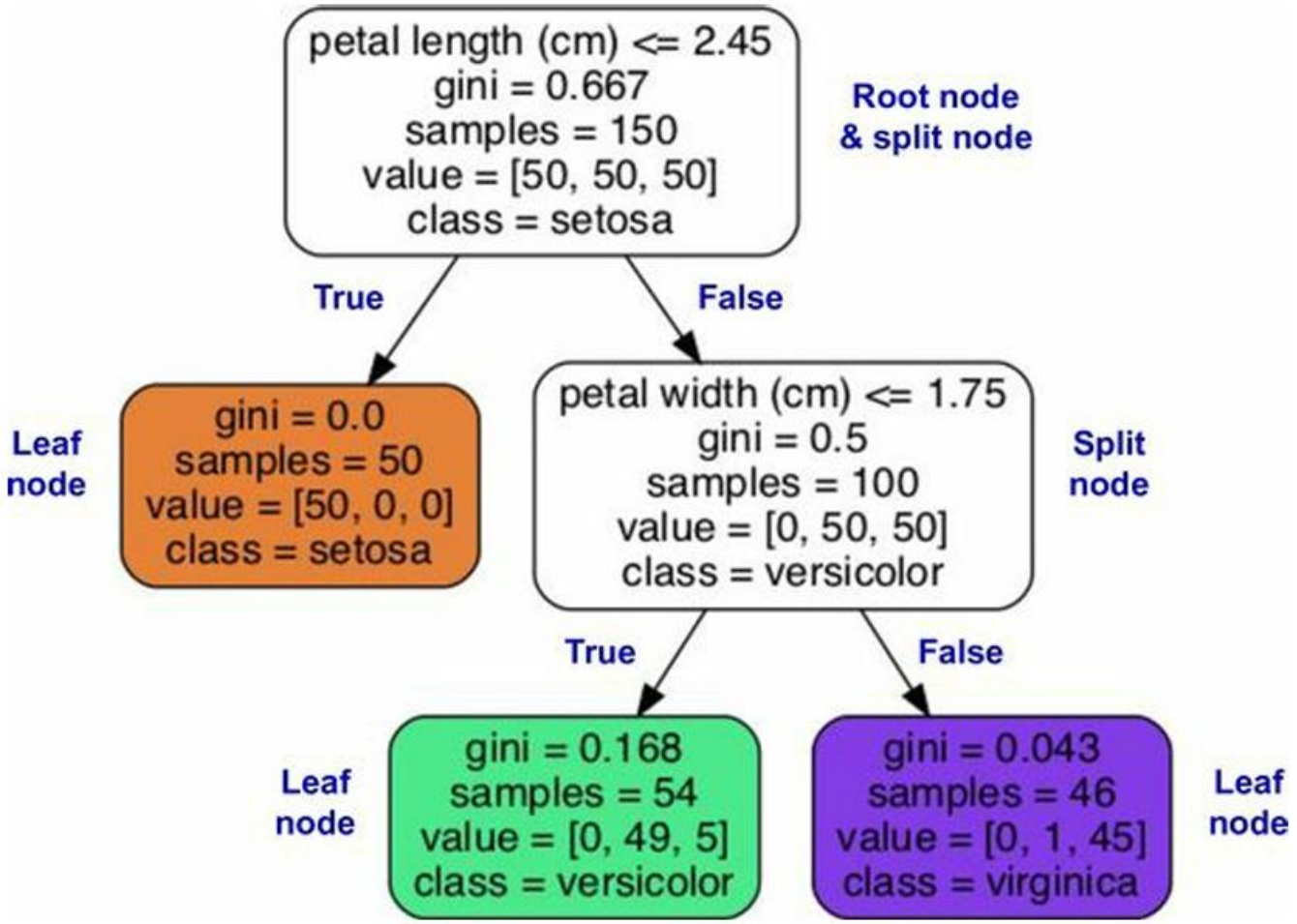
© 2025 Mohsen Zare. All rights reserved.

سپس با استفاده از graphviz.Source.from_file() می‌توان فایل *dot*. تولید شده را مستقیماً در یک Jupyter Notebook بارگذاری و نمایش داد. این روش امکان مشاهده‌ی تصویری درخت تصمیم، شامل گره‌ها، شاخه‌ها و معیارهای تصمیم‌گیری را فراهم می‌کند:

```
from graphviz import Source

Source.from_file("iris_tree.dot")
```

Graphviz یک بسته نرم‌افزاری متن‌باز برای مصورسازی گراف‌هاست. این ابزار شامل برنامه‌ای به نام dot نیز می‌شود که می‌تواند فایل‌های *dot*. را به فرمت‌های مختلفی مانند PDF یا PNG تبدیل کند.

اولین درخت تصمیم شما به شکل ۶-۱ شبیه است.



شکل ۶-۱. درخت تصمیم گیری Iris

© 2025 Mohsen Zare. All rights reserved.

پیش‌بینی با درخت تصمیم

برای درک نحوه پیش‌بینی توسط درخت تصمیم نشان داده شده در شکل ۶-۱، فرض کنید می‌خواهید یک گل زنبق را بر اساس ویژگی‌های گلبرگ آن طبقه‌بندی کنید. روند پیش‌بینی به شرح زیر است:

۱. شروع از «گره ریشه ($Root\ Node$)»: مسیر پیش‌بینی از بالاترین گره درخت آغاز می‌شود (عمق ۰). این گره سؤال می‌کند: «آیا طول گلبرگ کمتر از ۲٫۴۵ سانتی‌متر است؟»

۲. انتخاب مسیر مناسب: اگر پاسخ مثبت باشد، مسیر به گره فرزند سمت چپ (عمق ۱) ادامه می‌یابد. این گره یک «گره برگ ($Leaf\ Node$)» است، یعنی هیچ فرزند دیگری ندارد و دیگر سؤالی مطرح نمی‌کند.

۳. اعلام کلاس پیش‌بینی‌شده: در گره برگ، کافی است کلاس پیش‌بینی‌شده را مشاهده کنید. در این مثال، درخت تصمیم گل را به عنوان $Iris\ setosa$ طبقه‌بندی می‌کند.

به این ترتیب، هر نمونه با دنبال کردن مسیر مشخص از ریشه تا برگ، به کلاس مناسب اختصاص می‌یابد. حال فرض کنید گل جدیدی پیدا کرده‌اید که طول گلبرگ آن بیشتر از ۲٫۴۵ سانتی‌متر است. روند پیش‌بینی به شکل زیر ادامه می‌یابد:

۱. شروع از ریشه: مانند قبل، مسیر پیش‌بینی از گره ریشه آغاز می‌شود. چون طول گلبرگ بیشتر است، مسیر به گره فرزند سمت راست (عمق ۱) هدایت می‌شود.

۲. «گره شکافته ($Split\ Node$)»: این گره برگ نیست و سوال جدیدی مطرح می‌کند: «آیا عرض گلبرگ کمتر از ۱٫۷۵ سانتی‌متر است؟»

۳. انتخاب مسیر برگ مناسب:

- اگر پاسخ مثبت باشد، نمونه به گره برگ سمت چپ (عمق ۲) می‌رود و پیش‌بینی درخت، گل را به عنوان $Iris\ versicolor$ تعیین می‌کند.

© 2025 Mohsen Zare. All rights reserved.

- اگر پاسخ منفی باشد، نمونه به گره برگ سمت راست (عمق ۲) می‌رود و کلاس پیش‌بینی‌شده *Iris virginica* خواهد بود.

به این ترتیب، درخت تصمیم با پرسیدن مجموعه‌ای از سؤالات متوالی، نمونه‌ها را به سادگی و به صورت دقیق دسته‌بندی می‌کند.

> **نکته**
>
> یکی از ویژگی‌های فراوان درخت‌های تصمیم‌گیری این است که به عملیات‌های آماده‌سازی داده‌ی بسیار کمی نیاز دارند. در واقع، آنها اصلاً نیازی به مقیاس‌بندی(scaling) یا مرکزی‌سازی(centering) ویژگی‌ها ندارند.

ویژگی samples در هر گره، نشان‌دهنده تعداد نمونه‌های آموزشی‌ای است که به آن گره وارد می‌شوند. برای مثال، در گره عمق ۱ سمت راست، ۱۰۰ نمونه دارای طول گلبرگی بزرگ‌تر از ۲٫۴۵ سانتی‌متر هستند. از میان این ۱۰۰ نمونه، ۵۴ نمونه عرض گلبرگی کمتر از ۱٫۷۵ سانتی‌متر دارند و در نتیجه به گره عمق ۲ سمت چپ منتقل می‌شوند. ویژگی value توزیع کلاس‌ها را در یک گره نشان می‌دهد؛ یعنی تعداد نمونه‌هایی که از هر کلاس به آن گره رسیده‌اند. به عنوان نمونه، در گره پایینِ سمت راست مقدار value برابر $[0, 1, 45]$ است؛ یعنی هیچ نمونه‌ای از کلاس *Iris setosa*، یک نمونه از *Iris versicolor* و ۴۵ نمونه از *Iris virginica* در این گره قرار دارند. ویژگی gini میزان «ناخالصی جینی (*Gini impurity*)» یک گره را نشان می‌دهد(معیاری برای انتخاب ویژگی‌ها در درخت تصمیم؛ هرچه مقدار کمتر باشد، گره "خالص‌تر" است–یعنی داده‌های آن بیشتر متعلق به یک کلاس هستند). یک گره زمانی «خالص» است که همه نمونه‌های آن به یک کلاس تعلق داشته باشند، در این حالت مقدار gini برابر صفر است. برای مثال، گره سمت چپ در عمق ۱ فقط شامل نمونه‌های *Iris setosa* است، بنابراین خالص بوده و gini آن صفر است. بر اساس معادله ۶-۱، ناخالصی جینی G_i گره i اُم از طریق جمع مربعات نسبت نمونه‌های هر کلاس و کم‌کردن آن از ۱ محاسبه می‌شود. برای نمونه، گره سمت چپ در عمق ۲ دارای توزیع $[0, 49, 5]$ است و ناخالصی Gini آن تقریباً برابر با $1-(0/54)^2-(49/54)^2-(5/54)^2 \approx 0.168$ محاسبه می‌شود.

© 2025 Mohsen Zare. All rights reserved.

معادله ۶-۱. ناخالصی Gini

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

در این معادله:

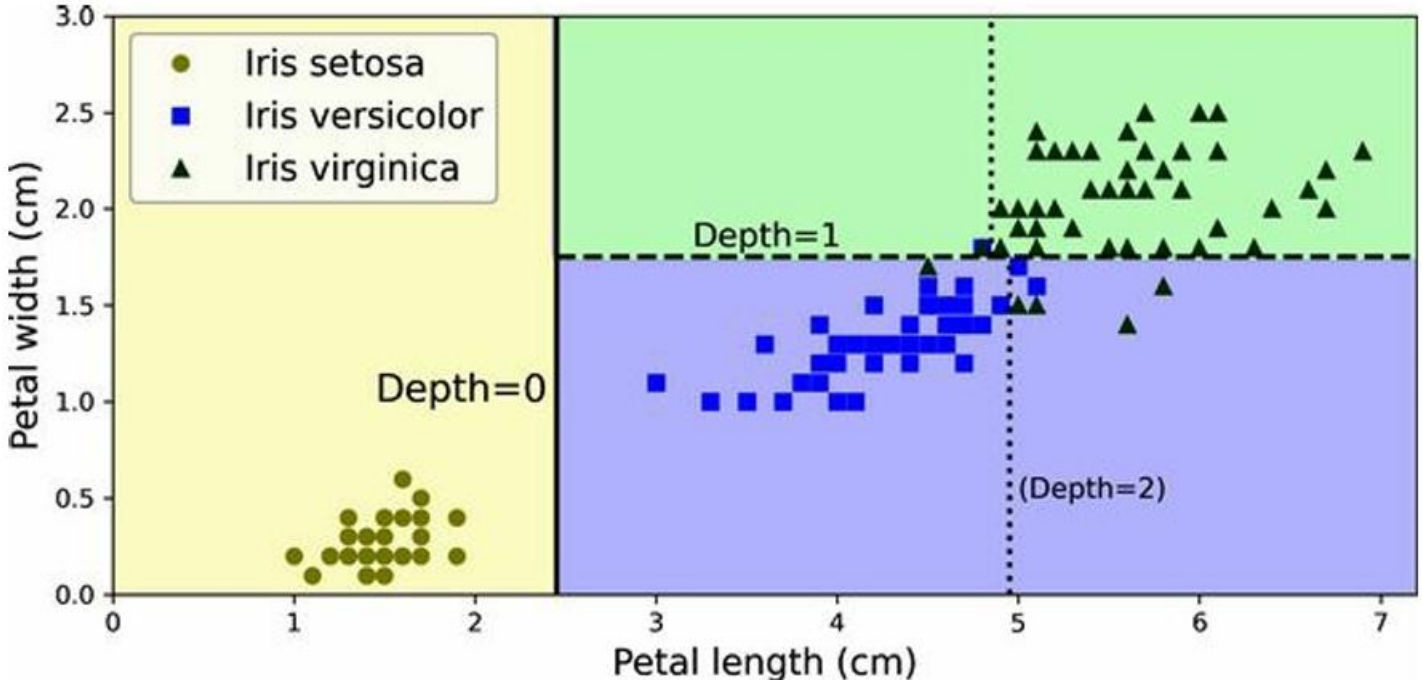
- G_i نشان‌دهنده‌ی ناخالصی Gini در گره i اُم است.
- $p_{i,k}$ بیانگر نسبت نمونه‌های کلاس k در میان تمام نمونه‌های آموزشی موجود در گره i اُم است.

نکته

تابخانه‌ی Scikit-Learn برای ساخت درخت‌های تصمیم از الگوریتم CART استفاده می‌کند. این الگوریتم فقط «درخت‌های باینری (*binary trees*)» ایجاد می‌کند؛ به این معنا که هر گره تقسیم‌شده دقیقاً دو فرزند دارد و پرسش‌های آن به‌صورت «بله/خیر» مطرح می‌شوند. با این حال، الگوریتم‌های دیگری مانند ID3 وجود دارند که می‌توانند درخت‌های تصمیمی با گره‌هایی بیش از دو فرزند تولید کنند.

شکل ۶-۲ مرزهای تصمیم‌گیری درخت تصمیم آموزش‌داده‌شده را نشان می‌دهد. خط عمودی پررنگ، مرز تصمیم گره ریشه (عمق ۰) را مشخص می‌کند: طول گلبرگ = ۲٫۴۵ سانتی‌متر. از آنجا که ناحیه‌ی سمت چپ کاملاً خالص است (فقط شامل نمونه‌های *Iris setosa*)، امکان تقسیم بیشتر آن وجود ندارد. اما ناحیه‌ی سمت راست همچنان ناخالص باقی مانده است، بنابراین گره سمت راست در عمق ۱، آن را در نقطه‌ی عرض گلبرگ = ۱٫۷۵ سانتی‌متر (نشان داده‌شده با خط‌چین) تقسیم می‌کند. از آنجا که مقدار max_depth روی ۲ تنظیم شده بود، درخت در همین سطح متوقف می‌شود. اگر مقدار max_depth برابر ۳ در نظر گرفته شود، دو گره در عمق ۲ هر کدام یک مرز تصمیم‌گیری اضافی ایجاد می‌کنند (که با دو خط‌چین عمودی مشخص شده‌اند).

© 2025 Mohsen Zare. All rights reserved.



شکل ۶-۲. مرزهای تصمیم در درخت تصمیم

نکته

ساختار درخت، همراه با تمام اطلاعات نشان‌داده‌شده در شکل ۶-۱، از طریق ویژگی tree_ attribute در آبجکت طبقه‌بند در دسترس است. برای مشاهده جزئیات بیشتر می‌توانید دستور **help(tree_clf.tree_)** را اجرا کنید. همچنین برای نمونه‌های عملی، به نوت‌بوک راهنمای این فصل مراجعه کنید.

© 2025 Mohsen Zare. All rights reserved.

تفسیر مدل: جعبه سفید در مقابل جعبه سیاه

درخت‌های تصمیم‌گیری مدل‌هایی شهودی اند و به‌راحتی قابل تفسیر هستند. به همین دلیل، چنین مدل‌هایی اغلب «مدل‌های جعبه سفید ($white\ box\ models$)» نامیده می‌شوند. در مقابل، الگوریتم‌هایی مانند جنگل‌های تصادفی و شبکه‌های عصبی معمولاً «مدل‌های جعبه سیاه ($black\ box\ models$)» محسوب می‌شوند. این مدل‌ها پیش‌بینی‌های دقیقی ارائه می‌دهند و می‌توان محاسبات انجام‌شده برای رسیدن به پیش‌بینی را بررسی کرد، اما توضیح ساده‌ای درباره‌ی دلیل وقوع یک پیش‌بینی خاص معمولاً دشوار است. برای مثال، اگر یک شبکه عصبی پیش‌بینی کند که یک فرد خاص در تصویر حضور دارد، مشخص نیست که کدام ویژگی‌ها نقش داشته‌اند: چشم‌ها؟ دهان؟ بینی؟ کفش‌ها؟ یا حتی مبلی که روی آن نشسته بود؟ در مقابل، درخت‌های تصمیم‌گیری قوانین طبقه‌بندی واضح و ساده‌ای ارائه می‌دهند که حتی می‌توان به صورت دستی نیز از آنها استفاده کرد، مانند طبقه‌بندی گل‌ها. حوزه‌ی «یادگیری ماشینی قابل تفسیر ($interpretable\ ML$)»، به دنبال توسعه سیستم‌هایی است که بتوانند تصمیمات خود را به روشی روشن و قابل فهم برای انسان توضیح دهند. این قابلیت در بسیاری از کاربردها اهمیت دارد، به‌ویژه برای اطمینان از اینکه سیستم‌ها تصمیمات ناعادلانه یا نامطلوب اتخاذ نمی‌کنند.

© 2025 Mohsen Zare. All rights reserved.

تخمین احتمالات کلاس

یک درخت تصمیم همچنین می‌تواند احتمال تعلق یک نمونه به کلاس خاصِ k را نیز تخمین بزند. ابتدا، درخت نمونه را پیمایش می‌کند تا به گره برگ مربوط برسد. سپس، نسبت نمونه‌های آموزشی متعلق به کلاس k در آن گره برگ را به عنوان احتمال برمی‌گرداند. برای مثال، فرض کنید گلی با طول گلبرگ ۵ سانتی‌متر و عرض ۱٫۵ سانتی‌متر پیدا کرده‌اید. گره برگ مربوط به این نمونه، گره سمت چپ با عمق ۲ است. در این حالت، درخت تصمیم احتمال تعلق نمونه به هر کلاس را به شکل زیر تخمین می‌زند: ۰% برای *Iris setosa* (۰ از ۵۴ نمونه) ، ۹۰٫۷ % برای *Iris versicolor* (۴۹ از ۵۴ نمونه) و ۹٫۳ % برای *Iris virginica* (۵ از ۵۴ نمونه). اگر بخواهیم کلاس پیش‌بینی‌شده را مشخص کنیم، درخت تصمیم *Iris versicolor* (کلاس ۱) را انتخاب می‌کند، زیرا بیشترین احتمال را دارد:

```
>>> tree_clf.predict_proba([[5, 1.5]]).round(3)
array([[0. , 0.907, 0.093]])
>>> tree_clf.predict([[5, 1.5]])
array([1])
```

توجه داشته باشید که احتمالات برآوردشده در هر گره برگ مشابه هستند. به عنوان مثال، اگر گلی با طول گلبرگ ۶ سانتی‌متر و عرض ۱٫۵ سانتی‌متر داشته باشیم، در گره برگ متناظر همان نسبت‌های کلاس محاسبه می‌شوند، حتی اگر به‌طور واضح مشخص باشد که این نمونه به احتمال زیاد *Iris virginica* است.

© 2025 Mohsen Zare. All rights reserved.

الگوریتم آموزش CART

Scikit-Learn برای آموزش درخت‌های تصمیم‌گیری، از الگوریتم «درخت طبقه‌بندی و رگرسیون (CART: *Classification and Regression Tree*)» استفاده می‌کند، که گاهی به آنها درخت‌های «رشدکننده» نیز گفته می‌شود. این الگوریتم با تقسیم مجموعه آموزشی به دو زیرمجموعه بر اساس یک ویژگی خاص k و یک آستانه t_k شروع می‌کند؛ برای مثال: «طول گلبرگ ≥ ۲٫۴۵ سانتی‌متر». اما چگونه k و t_k انتخاب می‌شوند؟ الگوریتم به دنبال زوجِ (k, t_k) می‌گردد که خالص‌ترین زیرمجموعه‌ها را تولید کند، به طوری که این خلوص با وزن اندازه هر زیرمجموعه لحاظ می‌شود. معادله ۶-۲، تابع هزینه‌ای را نشان می‌دهد که الگوریتم سعی در مینیمم‌سازی آن دارد.

معادله ۶-۲. تابع هزینه CART برای طبقه‌بندی

$$J(k, t_k) = G_{\text{left}} \frac{m_{\text{left}}}{m} + G_{\text{right}} \frac{m_{\text{right}}}{m}$$

که در آن:

- m_{left} , m_{right} ⟵ تعداد نمونه‌ها در زیرمجموعه چپ و راست
- $m = m_{\text{left}} + m_{\text{right}}$ ⟵ تعداد کل نمونه‌ها در گره
- G_{left} , G_{right} ⟵ معیار ناخالصی (مثل Gini Impurity یا Entropy) در زیرمجموعه چپ و راست

پس از اینکه الگوریتم CART موفق شد مجموعه آموزشی را به دو زیرمجموعه تقسیم کند، هر زیرمجموعه به صورت بازگشتی با همان منطق تقسیم می‌شود و این روند برای زیرمجموعه‌ها و به همین ترتیب ادامه می‌یابد. این فرآیند تا زمانی ادامه پیدا می‌کند که یکی از شرایط توقف رخ دهد: رسیدن به حداکثر عمق درخت (max_depth) یا عدم توانایی در یافتن تقسیم‌بندی که ناخالصی را کاهش دهد. علاوه بر این، چند هایپرپارامتر دیگر وجود دارند که شرایط توقف اضافی را تعیین می‌کنند، از جمله:

© 2025 Mohsen Zare. All rights reserved.

- min_samples_split
- min_samples_leaf
- min_weight_fraction_leaf
- max_leaf_nodes

هشدار

همان‌طور که مشاهده می‌کنید، الگوریتم CART یک «الگوریتم حریصانه (*greedy algorithm*)»– الگوریتمی که تصمیمات مرحله به مرحله می‌گیرد و همیشه بهترین گزینه محلی را انتخاب می‌کند؛ ممکن است بهترین راه‌حل سراسری را تضمین نکند–است: در هر سطح ابتدا به دنبال یافتن تقسیم‌بندی بهینه می‌گردد و سپس این روند را برای هر سطح بعدی به صورت بازگشتی تکرار می‌کند. الگوریتم بررسی نمی‌کند که آیا این تقسیم‌بندی در چند سطح پایین‌تر منجر به ناخالصی حداقل ممکن خواهد شد یا خیر. یک الگوریتم حریصانه، معمولاً راه‌حلی نسبتاً خوب ارائه می‌دهد، اما تضمینی برای بهینه بودن آن وجود ندارد.

متأسفانه، پیدا کردن درخت تصمیم بهینه یک مسئله *NP-complete* است که حل آن به زمانی در حدود $O(exp(m))$ نیاز دارد و حتی برای مجموعه‌های آموزشی کوچک نیز غیرقابل حل می‌شود. به همین دلیل، هنگام آموزش درخت‌های تصمیم، معمولا به یک راه‌حل نسبتاً خوب بسنده می‌کنیم.

© 2025 Mohsen Zare. All rights reserved.

پیچیدگی محاسباتی

انجام یک پیش‌بینی در درخت تصمیم نیازمند پیمایش از ریشه تا یک برگ است. از آنجا که درخت‌های تصمیم معمولاً تقریباً متعادل هستند، این پیمایش به طور میانگین مستلزم عبور از حدود $O(log_2(m))$ گره است؛ در اینجا $log_2(m)$ همان «binary logarithm یا لگاریتم باینری» m است که به صورت $log(m) / log(2)$ نیز بیان می‌شود. از آنجا که در هر گره فقط مقدار یک ویژگی بررسی می‌شود، پیچیدگی کل پیش‌بینی برابر با $O(log_2(m))$ است و این مقدار مستقل از تعداد ویژگی‌ها می‌باشد. بنابراین، پیش‌بینی در درخت‌های تصمیم حتی با مجموعه‌های آموزشی بسیار بزرگ نیز بسیار سریع است. در مقابل، الگوریتم آموزش در هر گره باید تمام ویژگی‌ها (یا تعداد کمتری اگر پارامتر max_features تنظیم شده باشد) را روی همه نمونه‌ها بررسی کند. در نتیجه، مقایسه‌ی تمام ویژگی‌ها روی تمام نمونه‌ها در هر گره منجر به پیچیدگی آموزشی در حدود $O(n \times mlog_2(m))$ می‌شود.

© 2025 Mohsen Zare. All rights reserved.

ناخالصی جینی یا آنتروپی؟

به طور پیش‌فرض، کلاس DecisionTreeClassifier در Scikit-Learn از معیار ناخالصی جینی (Gini impurity) استفاده می‌کند. با این حال، می‌توان با تنظیم هایپرپارامتر criterion="entropy"، معیار «آنتروپی (*entropy*)» را به عنوان معیار ناخالصی انتخاب کرد. مفهوم آنتروپی ریشه در ترمودینامیک دارد، جایی که به عنوان معیاری برای بی‌نظمی مولکولی تعریف می‌شود: هرچه مولکول‌ها ساکن‌تر و منظم‌تر باشند، مقدار آنتروپی به صفر نزدیک‌تر خواهد بود. این مفهوم بعدها در حوزه‌های مختلف گسترش یافت، از جمله در نظریه اطلاعات شانون (همان‌طور که در فصل ۴ دیدیم)، جایی که آنتروپی میانگین محتوای اطلاعاتی یک پیام را اندازه‌گیری می‌کند. اگر تمام پیام‌ها یکسان باشند، آنتروپی برابر صفر است. در یادگیری ماشین نیز آنتروپی به عنوان یک معیار ناخالصی استفاده می‌شود: زمانی که یک مجموعه تنها شامل نمونه‌هایی از یک کلاس باشد، مقدار آنتروپی آن مجموعه صفر خواهد بود. فرمول (معادله ۶-۳) آنتروپی برای گره i اُم را نشان می‌دهد. به عنوان نمونه، در گره سمت چپ با عمق ۲ در شکل ۶-۱، مقدار آنتروپی به صورت زیر محاسبه می‌شود:

$$-(49/54)log_2(49/54)-(5/54)log_2(5/54) \approx 0.445$$

معادله ۶-۳. آنتروپی

$$H_i = \sum_{k=1}^{n} p_{i,k} \;\; log_2(p_{i,k}) \;\; \text{for} \;\; p_{i,k} \neq 0$$

© 2025 Mohsen Zare. All rights reserved.

انتخاب بین ناخالصی جینی و آنتروپی

این پرسش طبیعی است که: کدام معیار بهتر است، ناخالصی جینی یا آنتروپی؟ واقعیت این است که در بیشتر موارد، تفاوت چندانی ایجاد نمی‌کنند و معمولاً هر دو منجر به درخت‌های بسیار مشابهی می‌شوند:

- ناخالصی جینی کمی سریع‌تر محاسبه می‌شود، بنابراین انتخاب پیش‌فرض مناسبی است.
- آنتروپی در مقایسه با جینی، تمایل دارد درخت‌هایی اندکی متعادل‌تر ایجاد کند.
- زمانی که نتایج متفاوت باشند، معمولاً جینی رایج‌ترین کلاس را در یک شاخه از درخت بهتر ایزوله می‌کند، در حالی که آنتروپی بیشتر به دنبال توزیع یکنواخت‌تر نمونه‌ها میان شاخه‌ها است.

به همین دلیل، در عمل انتخاب هر دو معیار قابل قبول است و تفاوت عمده‌ای در کارایی نهایی مدل ایجاد نمی‌کند.

© 2025 Mohsen Zare. All rights reserved.

هایپرپارامترهای منظم‌سازی

درخت‌های تصمیم بر خلاف مدل‌های خطی، فرضیات سختگیرانه‌ای درباره داده‌های آموزشی ندارند. برای مثال، مدل‌های خطی معمولاً فرض می‌کنند که رابطه بین ویژگی‌ها و برچسب‌ها خطی است، اما درخت‌های تصمیم هیچ فرض خاصی تحمیل نمی‌کنند. به همین دلیل، اگر هیچ محدودیتی برای آنها در نظر گرفته نشود، درخت به‌گونه‌ای رشد می‌کند که تقریباً تمام جزئیات داده‌های آموزشی را دنبال کند. این مسئله معمولاً منجر به بیش‌برازش می‌شود. به چنین مدلی اصطلاحاً «مدلِ غیرپارامتری (*nonparametric model*)» گفته می‌شود. این عنوان به این معنا نیست که مدل هیچ پارامتری ندارد، بلکه به این دلیل است که تعداد پارامترها پیش از آموزش مشخص نیست. در واقع، تعداد پارامترها می‌تواند همراه با داده‌ها رشد کند و ساختار درخت متناسب با داده‌ها تغییر یابد. در مقابل، یک «مدل پارامتری (*parametric model*)»–مانند رگرسیون خطی–تعداد پارامترهای ثابتی دارد که از پیش تعیین شده‌اند. این محدودیت باعث می‌شود چنین مدل‌هایی آزادی عمل کمتری داشته باشند، بنابراین احتمال ریسک بیش‌برازش در آنها کمتر است، هرچند ممکن است به قیمت کم‌برازش تمام شود.

برای جلوگیری از بیش‌برازش در درخت‌های تصمیم، باید میزان آزادی آنها را در طول آموزش محدود کرد؛ این فرآیند همان «منظم‌سازی یا *regularization*» است. نوع و تعداد هایپرپارامترهای منظم‌سازی به الگوریتم درخت بستگی دارد، اما تقریباً همیشه می‌توان دست‌کم حداکثر عمق درخت را کنترل کرد. در Scikit-Learn، این کار توسط هایپرپارامتر max_depth انجام می‌شود:

- مقدار پیش‌فرض آن None است، به این معنی که عمق درخت نامحدود خواهد بود.
- کاهش مقدار max_depth باعث محدود شدن رشد درخت می‌شود، در نتیجه مدل منظم‌تر خواهد شد و خطر بیش‌برازش کاهش می‌یابد.

کلاس DecisionTreeClassifier چند هایپرپارامتر دیگر نیز دارد که به شکل درخت تصمیم محدودیت اعمال می‌کنند:

- *max_features*: حداکثر تعداد ویژگی‌هایی که در هر گره برای تقسیم بررسی می‌شوند.

© 2025 Mohsen Zare. All rights reserved.

- *max_leaf_nodes*: بیشینه تعداد گره‌های برگ مجاز در درخت.
- *min_samples_split*: حداقل تعداد نمونه‌هایی که باید در یک گره وجود داشته باشند تا امکان تقسیم آن فراهم شود.
- *min_samples_leaf*: حداقل تعداد نمونه‌هایی که هر گره برگ باید شامل شود.
- *min_weight_fraction_leaf*: مشابه min_samples_leaf است، با این تفاوت که بر اساس کسری از وزن کل نمونه‌ها تعریف می‌شود.

به طور کلی، افزایش هایپرپارامترهای min_* یا کاهش هایپرپارامترهای max_* موجب منظم‌تر شدن مدل و کاهش خطر بیش‌برازش می‌شود.

نکته

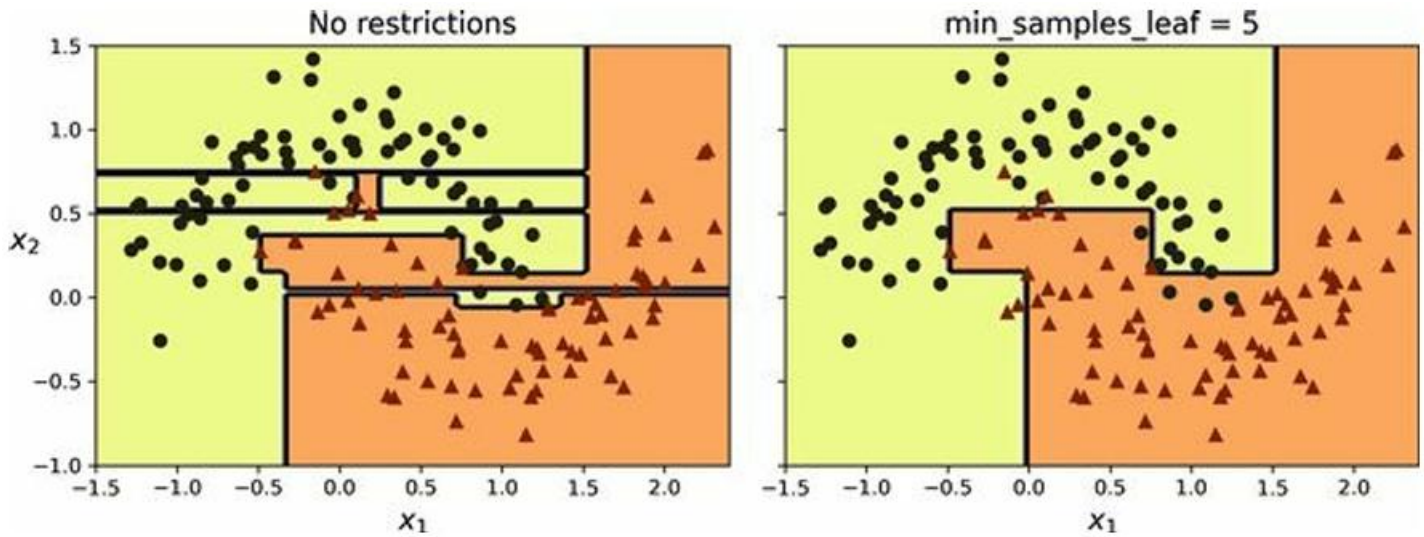
برخی الگوریتم‌های دیگر ابتدا درخت تصمیم را بدون هیچ محدودیتی آموزش می‌دهند و سپس گره‌های غیرضروری را از طریق «هرس (*pruning*)» حذف می‌کنند. در این روش، اگر گره‌ای تنها شامل فرزندان برگ باشد و بهبود خلوصی که ایجاد می‌کند از نظر آماری معنادار نباشد، آن گره غیرضروری تلقی می‌شود. برای ارزیابی این موضوع، از آزمون‌های آماری استاندارد مانند آزمون χ^2 (کایِ مربع یا chi-squared test) استفاده می‌شود تا احتمال آن‌که این بهبود صرفاً ناشی از شانس باشد («فرضیه صفر یا *null hypothesis*») برآورد گردد. اگر مقدار «p-value» از یک آستانه مشخص (معمولاً ۵٪ که توسط یک هایپرپارامتر تعیین می‌شود) فراتر رود، آن گره غیرضروری تشخیص داده می‌شود و فرزندانش حذف می‌شوند. این فرآیند هرس ادامه می‌یابد تا زمانی که همه گره‌های غیرضروری از درخت حذف شوند.

بیایید منظم‌سازی را روی مجموعه‌داده‌ی moons dataset (که در فصل ۵ معرفی شد) امتحان کنیم. در این آزمایش، ابتدا یک درخت تصمیم بدون هیچ‌گونه منظم‌سازی آموزش می‌دهیم، سپس همان مدل را با اعمال محدودیت min_samples_leaf=5 اجرا می‌کنیم. کد مربوطه به شکل زیر است و در شکل ۶-۳ مرزهای تصمیم‌گیری حاصل از هر دو درخت نمایش داده شده‌اند:

```
from sklearn.datasets import make_moons
```

© 2025 Mohsen Zare. All rights reserved.

```
X_moons, y_moons = make_moons(n_samples=150, noise=0.2, random_state=42)

tree_clf1 = DecisionTreeClassifier(random_state=42)
tree_clf2 = DecisionTreeClassifier(min_samples_leaf=5, random_state=42)
tree_clf1.fit(X_moons, y_moons)
tree_clf2.fit(X_moons, y_moons)
```



شکل ۶-۳. مرزهای تصمیم‌گیری یک درخت نامنظم (چپ) و یک درخت منظم (راست)

در شکل سمت چپ، مدل نامنظم به‌وضوح دچار بیش‌برازش شده است، در حالی‌که در شکل سمت راست، مدل منظم به‌احتمال زیاد تعمیم‌پذیری بهتری دارد. برای اطمینان از این موضوع، می‌توان هر دو درخت را روی یک مجموعه‌ی آزمایشی جدید (که با یک مقدار اولیه تصادفی یا random seed متفاوت تولید شده است—عددی که باعث می‌شود تولید اعداد تصادفی قابل تکرار و بازتولید باشد؛ برای مثال هنگام تقسیم داده‌ها به train/test یا راه‌اندازی الگوریتم‌ها) ارزیابی کرد تا عملکردشان مقایسه شود:

```
>>> X_moons_test, y_moons_test = make_moons(n_samples=1000, noise=0.2,
…                                 random_state=43)
>>> tree_clf1.score(X_moons_test, y_moons_test)
0.898
>>> tree_clf2.score(X_moons_test, y_moons_test)
0.92
```

در واقع، درخت دوم روی مجموعه‌ی آزمون عملکرد بهتری نشان می‌دهد و دقت بیشتری دارد.

© 2025 Mohsen Zare. All rights reserved.

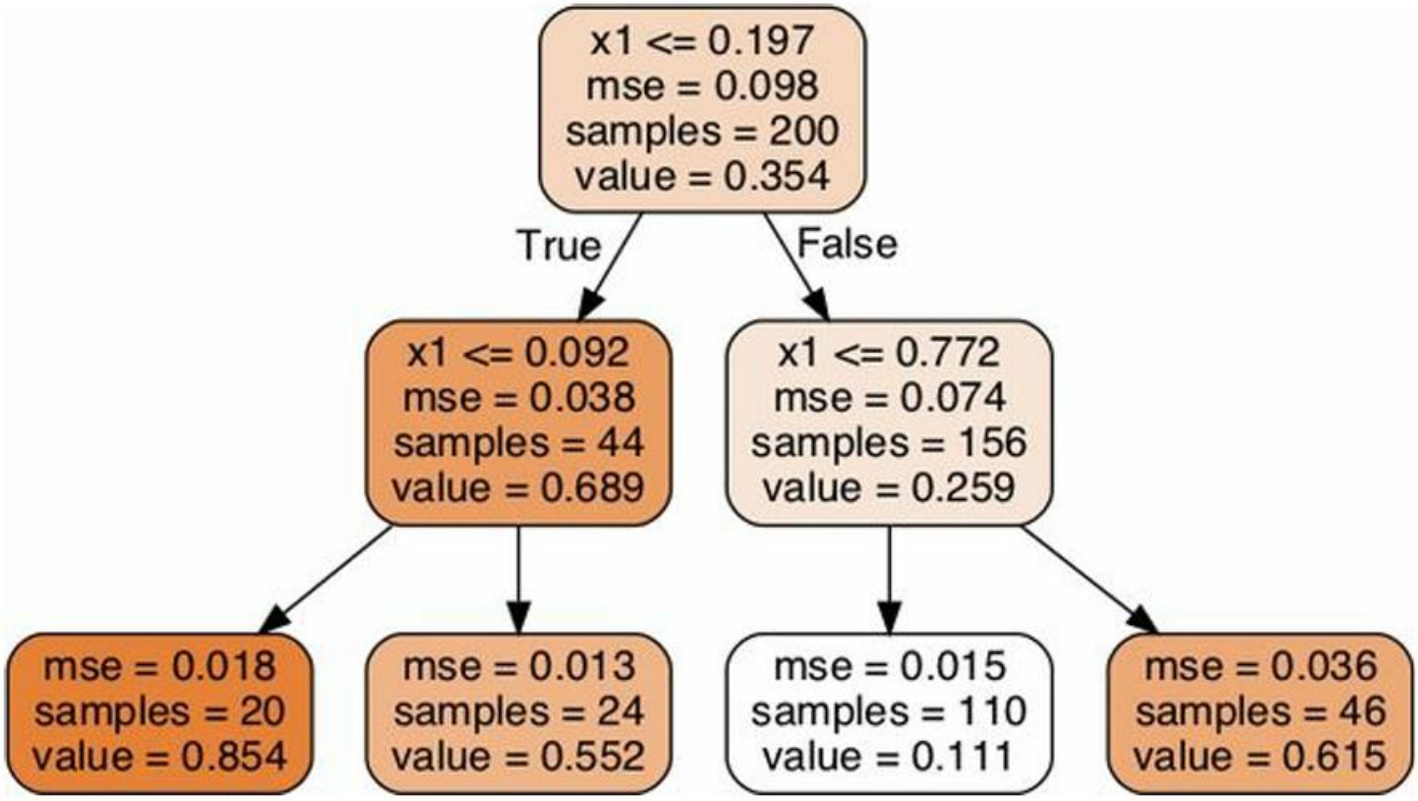
رگرسیون

درخت‌های تصمیم همچنین می‌توانند برای مسائل رگرسیون به کار روند. برای نمونه، می‌توان یک درخت رگرسیون با استفاده از کلاس DecisionTreeRegressor در Scikit-Learn ایجاد کرد و آن را روی یک مجموعه داده درجه دوم با نویز و با حداکثر عمق ۲ (max_depth=2) آموزش داد:

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

np.random.seed(42)
X_quad = np.random.rand(200, 1) - 0.5 # a single random input feature
y_quad = X_quad ** 2 + 0.025 * np.random.randn(200, 1)

tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X_quad, y_quad)
```

درخت حاصل در شکل ۶-۴ نشان داده شده است.

© 2025 Mohsen Zare. All rights reserved.

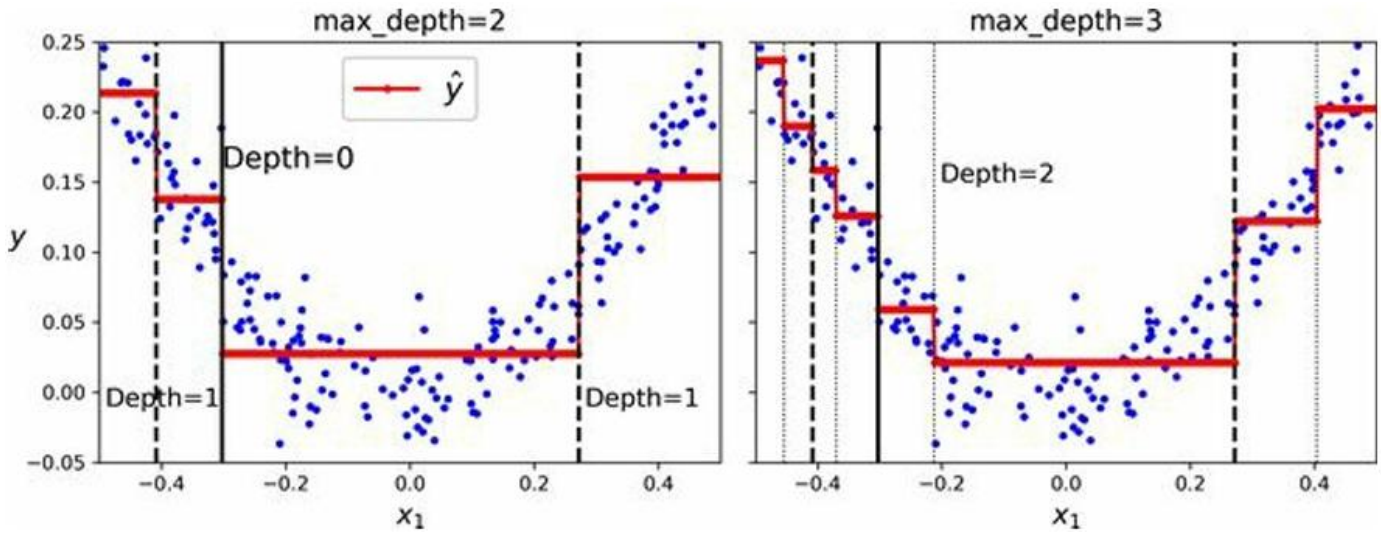


شکل ۶-۴. یک درخت تصمیم برای رگرسیون

این درخت بسیار شبیه درخت طبقه‌بندی ای است که پیش‌تر ایجاد کردیم. تفاوت اصلی آن در این است که به جای پیش‌بینی یک کلاس در هر گره، یک مقدار عددی را پیش‌بینی می‌کند. برای مثال، فرض کنید می‌خواهیم برای یک نمونه جدید با $x_1 = 0.2$ پیش‌بینی انجام دهیم. گره ریشه بررسی می‌کند که آیا $x_1 \leq 0.197$ هست یا خیر. از آنجا که شرط برقرار نیست، الگوریتم به گره فرزند سمت راست می‌رود که بررسی می‌کند آیا $x_1 \leq 0.772$ هست یا خیر. چون این شرط برقرار است، الگوریتم وارد گره فرزند چپ می‌شود که یک گره برگ است و مقدار ۰/۱۱۱ را پیش‌بینی می‌کند (value=0.111). این مقدار، میانگین مقادیر هدف ۱۱۰ نمونه آموزشی مرتبط با این گره برگ است و منجر به میانگین مربعات خطا (MSE) برابر با ۰/۰۱۵ روی این نمونه‌ها می‌شود.

پیش‌بینی‌های این مدل در سمت چپ شکل ۶-۵ نمایش داده شده‌اند. اگر max_depth=3 تنظیم شود، پیش‌بینی‌های نشان داده شده در سمت راست شکل را دریافت خواهید کرد. توجه داشته باشید که مقدار پیش‌بینی‌شده برای هر منطقه، همیشه میانگین مقادیر هدف نمونه‌های آن منطقه است. الگوریتم درخت تصمیم، هر ناحیه را به گونه‌ای تقسیم می‌کند که نمونه‌های آموزشی تا حد ممکن به مقدار پیش‌بینی‌شده نزدیک باشند.

© 2025 Mohsen Zare. All rights reserved.



شکل ۶-۵. پیش‌بینی‌های دو مدل رگرسیون درخت تصمیم

الگوریتم CART در رگرسیون نیز مشابه همان چیزی عمل می‌کند که پیش‌تر توضیح داده شد؛ با این تفاوت که این بار به جای تلاش برای تقسیم مجموعه آموزشی به شکلی که ناخالصی (impurity) مینیمم شود، هدف آن تقسیم داده‌ها به گونه‌ای است که میانگین مربعات خطا (MSE) کاهش یابد. به بیان دیگر، در هر گام الگوریتم سعی می‌کند بهترین ویژگی و بهترین آستانه را انتخاب کند تا داده‌های آموزشی به دو بخش تقسیم شوند و مقدار MSE حاصل از این تقسیم کمترین مقدار ممکن باشد. این ایده در قالب تابع هزینه‌ای که الگوریتم قصد بهینه‌سازی آن را دارد، در معادله ۶-۴ نمایش داده شده است.

معادله ۶-۴. تابع هزینه CART برای رگرسیون

$$J(k, t_k) = \frac{m_{\text{left}}}{m} MSE_{\text{left}} + \frac{m_{\text{right}}}{m} MSE_{\text{right}}$$

که در آن:

$$MSE_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} (y_i + \hat{y}_{\text{node}})^2$$

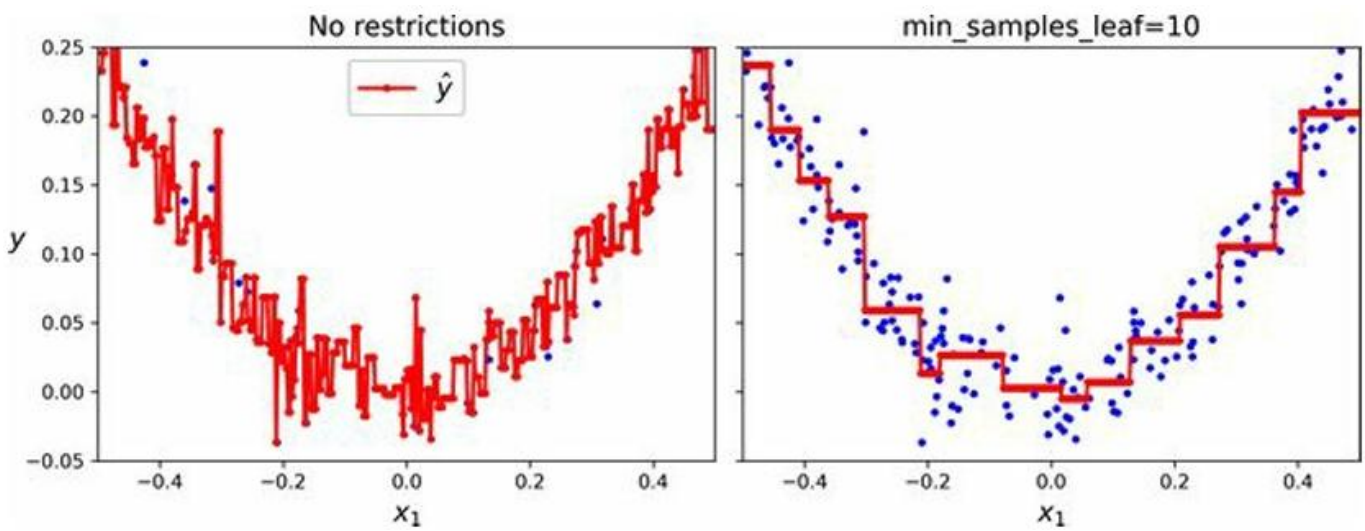
$$\hat{y}_{node} = \frac{1}{m_{node}} \sum_{i \in \text{node}} y_i$$

© 2025 Mohsen Zare. All rights reserved.

- m_{right} و m_{left} ⟵ تعداد نمونه‌ها در زیرمجموعه چپ و راست
- $m = m_{\text{left}} + m_{\text{right}}$ ⟵ تعداد کل نمونه‌ها
- $\hat{y}_{node}$ ⟵ مقدار میانگین پیش‌بینی در گره

دقیقاً مانند مسائل طبقه‌بندی، درخت‌های تصمیم در مسائل رگرسیون نیز به‌شدت در معرض بیش‌برازش قرار دارند. اگر هیچ نوع منظم‌سازی اعمال نشود (یعنی با استفاده از هایپرپارامترهای پیش‌فرض)، نتیجه همان پیش‌بینی‌هایی است که در سمت چپ شکل ۶-۶ مشاهده می‌کنید؛ مدلی که به‌طور افراطی با داده‌های آموزشی منطبق شده است.

با این حال، تنها با تنظیم ساده‌ی min_samples_leaf=10 می‌توان از این مشکل تا حد زیادی جلوگیری کرد. همان‌طور که در سمت راست شکل ۶-۶ نشان داده شده است، این تغییر باعث ایجاد مدلی معقول‌تر و تعمیم‌پذیرتر می‌شود.

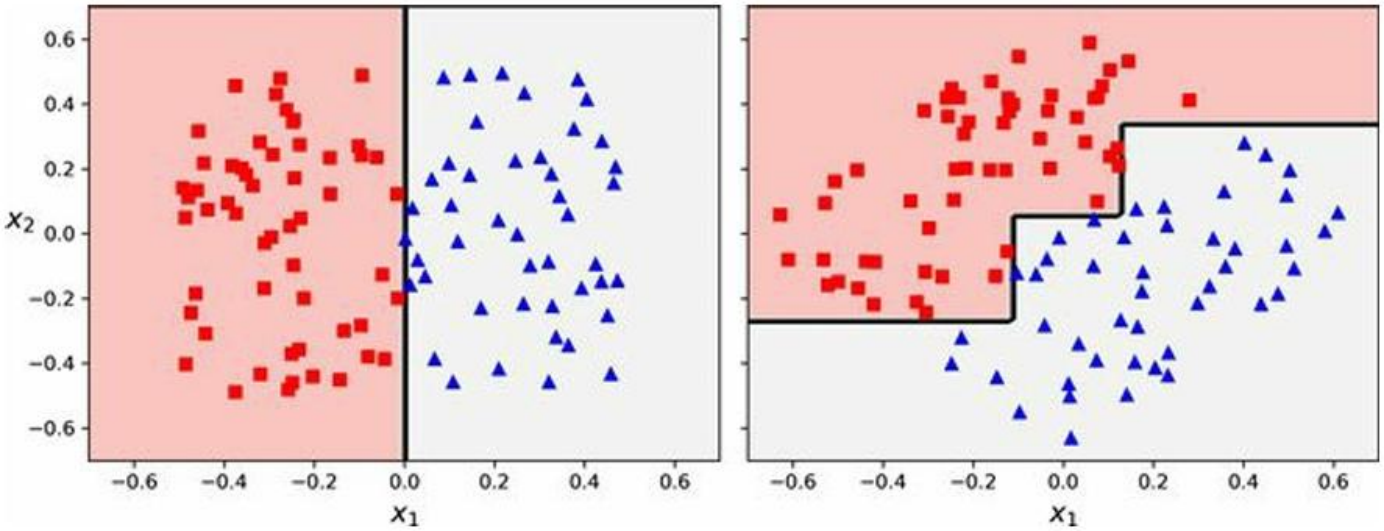


شکل ۶-۶. پیش‌بینی‌های یک درخت رگرسیون نامنظم (چپ) و یک درخت منظم (راست)

© 2025 Mohsen Zare. All rights reserved.

حساسیت به جهت محور

همان‌طور که تا اینجا دیدید، درخت‌های تصمیم مزایای زیادی دارند: تفسیر و درک آن‌ها آسان است، استفاده از آن‌ها ساده است و از نظر کاربردی هم بسیار قدرتمند محسوب می‌شوند. با این حال، این مدل‌ها بدون محدودیت نیستند. یکی از این محدودیت‌ها تمایل درخت‌های تصمیم به ایجاد مرزهای تصمیم‌گیری متعامد است؛ به این معنا که تمام تقسیم‌بندی‌ها عمود بر یکی از محورهای ویژگی انجام می‌شوند. همین موضوع باعث می‌شود که درخت‌های تصمیم به جهت‌گیری داده‌ها حساس باشند. برای نمونه، در شکل ۶-۷ یک مجموعه داده ساده با جدایش خطی نشان داده شده است. در سمت چپ، درخت تصمیم به‌راحتی داده‌ها را تقسیم می‌کند. اما وقتی همان داده‌ها ۴۵ درجه چرخانده شوند (سمت راست)، مرز تصمیم‌گیری به شکلی غیرضروری پیچیده درمی‌آید. هرچند هر دو درخت به‌خوبی با داده‌های آموزشی برازش دارند، ولی مدل سمت راست به احتمال زیاد توانایی تعمیم‌پذیری کمتری خواهد داشت.



شکل ۶-۷. حساسیت به چرخش مجموعه آموزشی

یک روش برای کاهش این مشکل، مقیاس‌بندی داده‌ها و سپس استفاده از «تحلیل مؤلفه‌های اصلی (*Principal Component Analysis* - PCA)» است. در فصل ۸ به‌طور کامل به PCA خواهیم پرداخت، اما فعلاً کافی است بدانید که این روش داده‌ها را به گونه‌ای می‌چرخاند که همبستگی بین

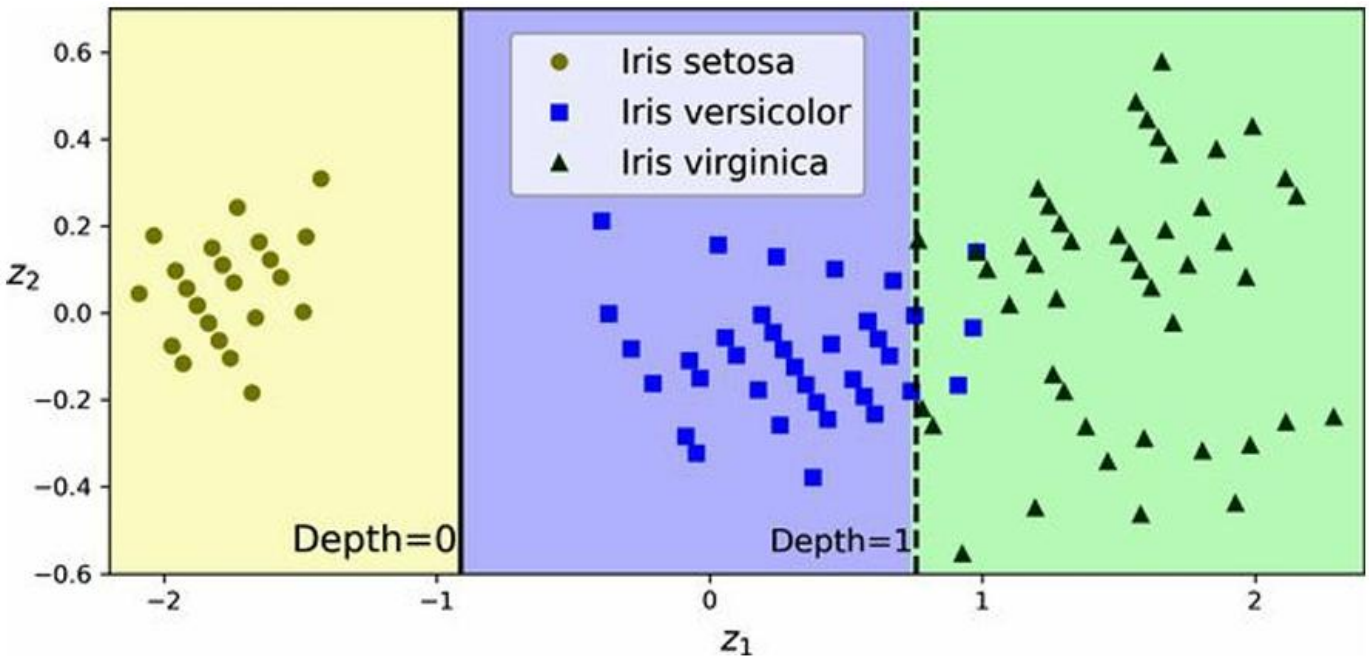
© 2025 Mohsen Zare. All rights reserved.

ویژگی‌ها کاهش یابد. این تغییر جهت اغلب (هرچند نه همیشه) باعث می‌شود که درخت تصمیم‌گیری راحت‌تر بتواند داده‌ها را مدل کند.

برای آزمایش این ایده، یک Pipeline ساده ایجاد می‌کنیم که ابتدا داده‌ها را مقیاس‌بندی کرده و سپس با استفاده از PCA آن‌ها را می‌چرخاند. در ادامه یک DecisionTreeClassifier روی این داده‌ها آموزش می‌دهیم. نتیجه در شکل ۶-۸ نمایش داده شده است: همان‌طور که می‌بینید، چرخش داده‌ها امکان برازش مناسب مجموعه داده را تنها با استفاده از یک ویژگی، یعنی z_1، فراهم می‌کند. این ویژگی ترکیبی خطی از طول و عرض گلبرگ اصلی است.

کد مربوطه به صورت زیر است:

```
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

pca_pipeline = make_pipeline(StandardScaler(), PCA())
X_iris_rotated = pca_pipeline.fit_transform(X_iris)
tree_clf_pca = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf_pca.fit(X_iris_rotated, y_iris)
```

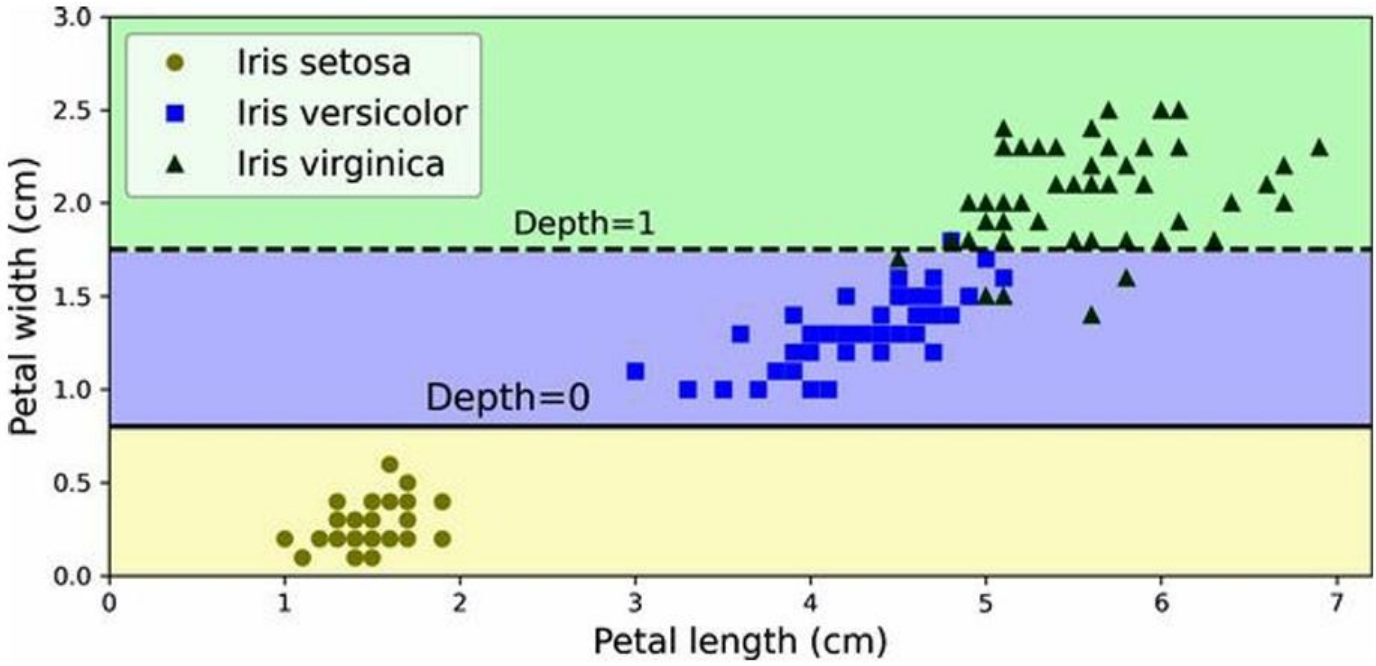


شکل ۶-۸. مرزهای تصمیم‌گیری یک درخت روی مجموعه داده ی زنبق مقیاس‌بندی شده و چرخش یافته با PCA

© 2025 Mohsen Zare. All rights reserved.

درخت‌های تصمیم واریانس بالایی دارند

به طور کلی، عیب اصلی درخت‌های تصمیم واریانس بالای آنهاست. حتی تغییرات جزئی در داده‌ها یا هایپرپارامترها می‌تواند منجر به ایجاد مدل‌هایی کاملاً متفاوت شود. دلیل دیگر این ناپایداری آن است که الگوریتم آموزش در Scikit-Learn ذاتاً تصادفی است؛ در هر گره، مجموعه‌ای تصادفی از ویژگی‌ها برای تقسیم بررسی می‌شود. بنابراین، حتی اگر همان داده‌ها را دوباره به همان مدل بدهید، ممکن است یک درخت تصمیم کاملاً متفاوت به دست آورید—مگر آنکه هایپرپارامتر random_state را ثابت نگه دارید. این موضوع در شکل ۶-۹ نشان داده شده است: همان داده‌ها دوباره آموزش داده شده‌اند، اما خروجی درخت تصمیم جدید به وضوح با درخت تصمیم قبلی (شکل ۶-۲) متفاوت است.



شکل ۶-۹. همانطور که می‌بینید، این درخت تصمیم با درخت تصمیم قبلی (شکل ۶-۲) بسیار متفاوت به نظر می‌رسد.

© 2025 Mohsen Zare. All rights reserved.

خوشبختانه، این مشکل واریانس بالا راه‌حل مؤثری دارد:

اگر به جای تکیه بر یک درخت منفرد، «مجموعه‌ای بزرگ از درخت‌ها ($ensemble\ of\ trees$)» را آموزش دهیم و سپس پیش‌بینی‌های آنها را میانگین‌گیری کنیم، نوسان‌ها به‌طور چشمگیری کاهش پیدا می‌کنند و مدل پایدارتر خواهد شد. به چنین مجموعه‌ای از درخت‌ها، «جنگل تصادفی ($random\ forest$)» گفته می‌شود. همان‌طور که در فصل بعد خواهید دید، جنگل‌های تصادفی جزو قدرتمندترین و پرکاربردترین مدل‌های یادگیری ماشین به شمار می‌آیند.

© 2025 Mohsen Zare. All rights reserved.

تمرین‌ها

۱. عمق تقریبی یک درخت تصمیم آموزش دیده (بدون محدودیت) روی یک مجموعه آموزشی با یک میلیون نمونه چقدر است؟

۲. آیا ناخالصی جینی یک گره معمولاً کمتر یا بیشتر از والدش است؟ آیا «معمولاً (*generally*)» کمتر یا بیشتر است، یا «همیشه(*always*)» کمتر یا بیشتر؟

۳. اگر یک درخت تصمیم، مجموعه آموزشی را بیش‌برازش می‌دهد، آیا ایده خوبی است که کاهش max_depth را امتحان کنیم؟

۴. اگر یک درخت تصمیم، مجموعه آموزشی را کمتر از حد برازش می‌دهد، آیا ایده خوبی است که مقیاس‌بندی ویژگی‌های ورودی را امتحان کنیم؟

۵. اگر آموزش یک درخت تصمیم روی یک مجموعه آموزشی شامل یک میلیون نمونه، یک ساعت طول بکشد، آموزش یک درخت تصمیم دیگر روی یک مجموعه آموزشی شامل ده میلیون نمونه تقریباً چقدر زمان خواهد برد؟ راهنمایی: پیچیدگی محاسباتی الگوریتم CART را در نظر بگیرید.

۶. اگر آموزش یک درخت تصمیم روی یک مجموعه آموزشی مشخص یک ساعت طول بکشد، اگر تعداد ویژگی‌ها را دو برابر کنید، تقریباً چقدر زمان لازم خواهد بود؟

۷. با دنبال کردن این مراحل، یک درخت تصمیم برای مجموعه داده‌ی moons آموزش داده و تنظیم کنید:

الف. برای تولید مجموعه داده‌ی ماه‌ها از make_moons(n_samples=10000, noise=0.4) استفاده کنید.

ب. از تابع train_test_split() برای تقسیم مجموعه داده‌ها به یک مجموعه آموزشی و یک مجموعه آزمایشی استفاده کنید.

ج. از جستجوی شبکه‌ای به همراه اعتبارسنجی متقاطع (با کمک کلاس GridSearchCV) برای یافتن مقادیر مناسب برای هایپرپارامتر یک DecisionTreeClassifier استفاده کنید. راهنمایی: مقادیر مختلفی را برای max_leaf_nodes امتحان کنید.

© 2025 Mohsen Zare. All rights reserved.

ح. با استفاده از این هایپرپارامترها، آن را روی کل مجموعه آموزشی آموزش دهید و عملکرد مدل خود را روی مجموعه آزمایشی اندازه‌گیری کنید. شما باید تقریباً ۸۵% تا ۸۷% دقت داشته باشید.

۸. با دنبال کردن این مراحل، یک جنگل پرورش دهید:

الف. در ادامه تمرین قبلی، ۱,۰۰۰ زیرمجموعه از مجموعه آموزشی ایجاد کنید که هر کدام شامل ۱۰۰ نمونه هستند که به صورت تصادفی انتخاب شده‌اند. راهنمایی: می‌توانید برای این کار از کلاس ShuffleSplit در Scikit-Learn استفاده کنید.

ب. با استفاده از بهترین مقادیر هایپرپارامتر یافت شده در تمرین قبلی، روی هر زیرمجموعه یک درخت تصمیم آموزش دهید. این ۱,۰۰۰ درخت تصمیم را روی مجموعه آزمون ارزیابی کنید. از آنجایی که آنها روی مجموعه‌های کوچکتری آموزش دیده‌اند، این درخت‌های تصمیم احتمالاً عملکرد بدتری نسبت به درخت تصمیم اول خواهند داشت و تنها به حدود ۸۰% دقت دست می‌یابند.

ج. حالا نوبت جادو می‌رسد. برای هر نمونه از مجموعه آزمون، پیش‌بینی‌های ۱۰۰۰ درخت تصمیم‌گیری را تولید کنید و فقط پیش‌بینی‌هایی را که بیشترین تکرار را دارند نگه دارید (می‌توانید از تابع mode() در SciPy برای این کار استفاده کنید). این رویکرد، پیش‌بینی‌های اکثریت آراء (*majority-vote predictions*) را در مجموعه آزمون به شما ارائه می‌دهد.

ح. این پیش‌بینی‌ها را روی مجموعه آزمون ارزیابی کنید: باید دقتی کمی بالاتر از مدل اول خود (حدود ۰/۵ تا ۱/۵ % بالاتر) به دست آورید. تبریک می‌گویم، شما یک طبقه‌بندی‌کننده جنگل تصادفی آموزش داده‌اید!

پاسخ‌های مربوط به این تمرین‌ها را می‌توانید در انتهای نوت‌بوک همین فصل، از طریق نشانی *https://homl.info/colab3* مشاهده کنید.

© 2025 Mohsen Zare. All rights reserved.

فصل ۷. یادگیری تجمیعی و جنگل‌های تصادفی

فرض کنید یک پرسش دشوار را از هزاران نفر به‌طور تصادفی بپرسید و سپس پاسخ‌های آن‌ها را ترکیب کنید. در بسیاری از موارد، نتیجه‌ی کلی به‌دست‌آمده دقیق‌تر از پاسخی است که یک متخصص ارائه می‌دهد. این پدیده را «خرد جمعی یا wisdom of the crowd» می‌نامند. به‌طور مشابه، اگر پیش‌بینی‌های یک گروه از مدل‌ها (مانند طبقه‌بندها یا رگرسیونرها) را با هم ترکیب کنید، معمولاً نتیجه‌ای به دست می‌آید که از بهترین مدل منفرد نیز بهتر است. چنین گروهی از مدل‌ها را «مجموعه یا گروه (*ensemble*)» می‌نامند. به همین دلیل، این رویکرد را «یادگیری تجمیعی یا ترکیبی (*ensemble learning*)» می‌گویند و الگوریتمی که آن را پیاده‌سازی می‌کند، یک «روش تجمیعی یا ترکیبی (*ensemble method*)» محسوب می‌شود.

به‌عنوان مثال یک روش ترکیبی، می‌توان مجموعه‌ای از طبقه‌بندهای درخت تصمیم را آموزش داد، به‌طوری که هر درخت روی یک زیرمجموعه تصادفی متفاوت از داده‌های آموزشی تمرین کند. سپس پیش‌بینی‌های تمام درخت‌ها جمع‌آوری می‌شوند و کلاسی که بیشترین رأی را کسب کند به‌عنوان پیش‌بینی نهایی گروه انتخاب می‌شود (برای جزئیات، به آخرین تمرین فصل ۶ مراجعه کنید). چنین

© 2025 Mohsen Zare. All rights reserved.

مجموعه‌ای از درخت‌های تصمیم، «جنگل تصادفی یا *random forest*» نامیده می‌شود و با وجود ساختار نسبتاً ساده‌اش، یکی از قدرتمندترین الگوریتم‌های یادگیری ماشین موجود است.

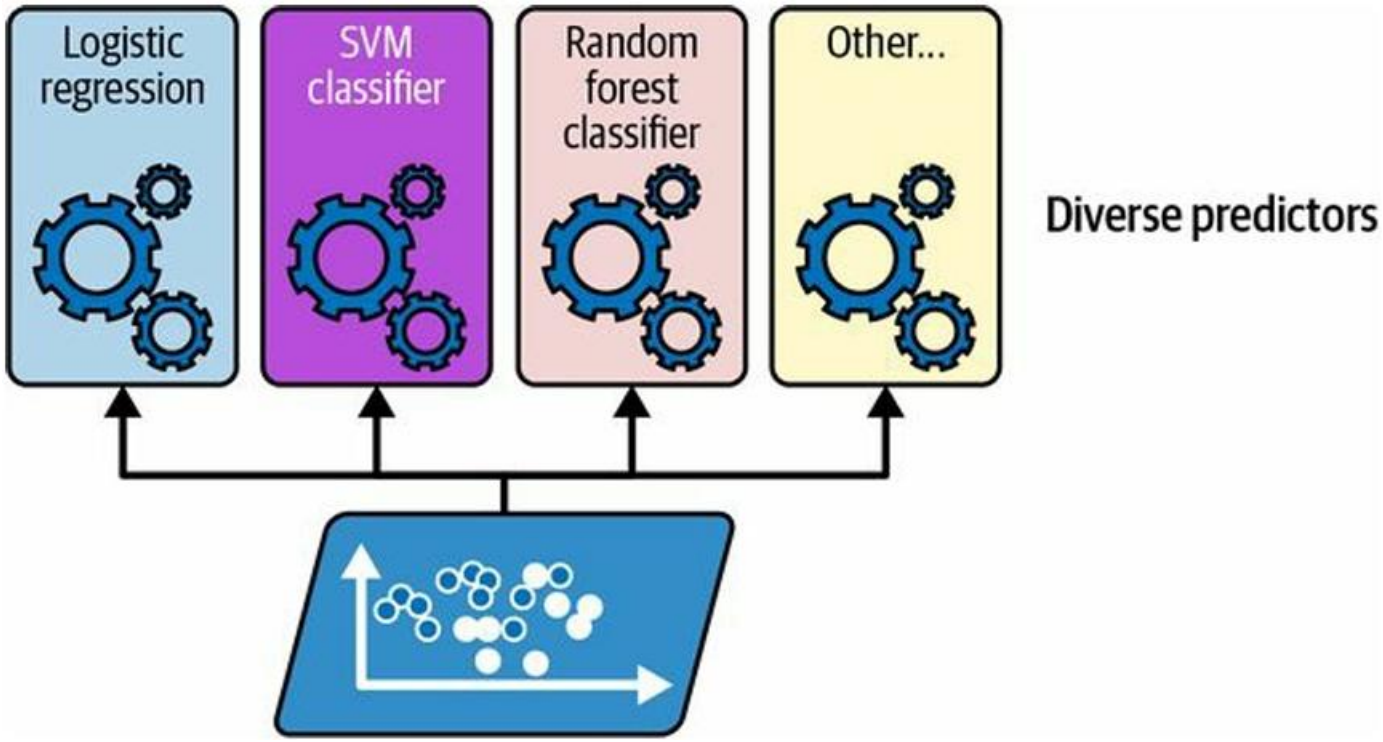
همان‌طور که در فصل ۲ اشاره شد، اغلب در مراحل پایانی یک پروژه، زمانی که چند پیش‌بینی‌کننده خوب ساخته شده‌اند، از روش‌های تجمیعی برای ترکیب آنها و ایجاد یک پیش‌بینی‌کننده قدرتمندتر استفاده می‌شود. در واقع، راه‌حل‌های برنده در مسابقات یادگیری ماشین معمولاً شامل چندین روش ترکیبی هستند؛ نمونه مشهور آن، مسابقه Netflix Prize است.

در این فصل، به بررسی محبوب‌ترین روش‌های گروهی می‌پردازیم، از جمله: voting classifiers، مجموعه‌های bagging و pasting، random forests، و تکنیک‌های تجمیعی boosting و stacking.

© 2025 Mohsen Zare. All rights reserved.

طبقه‌بندهای رأی‌گیری

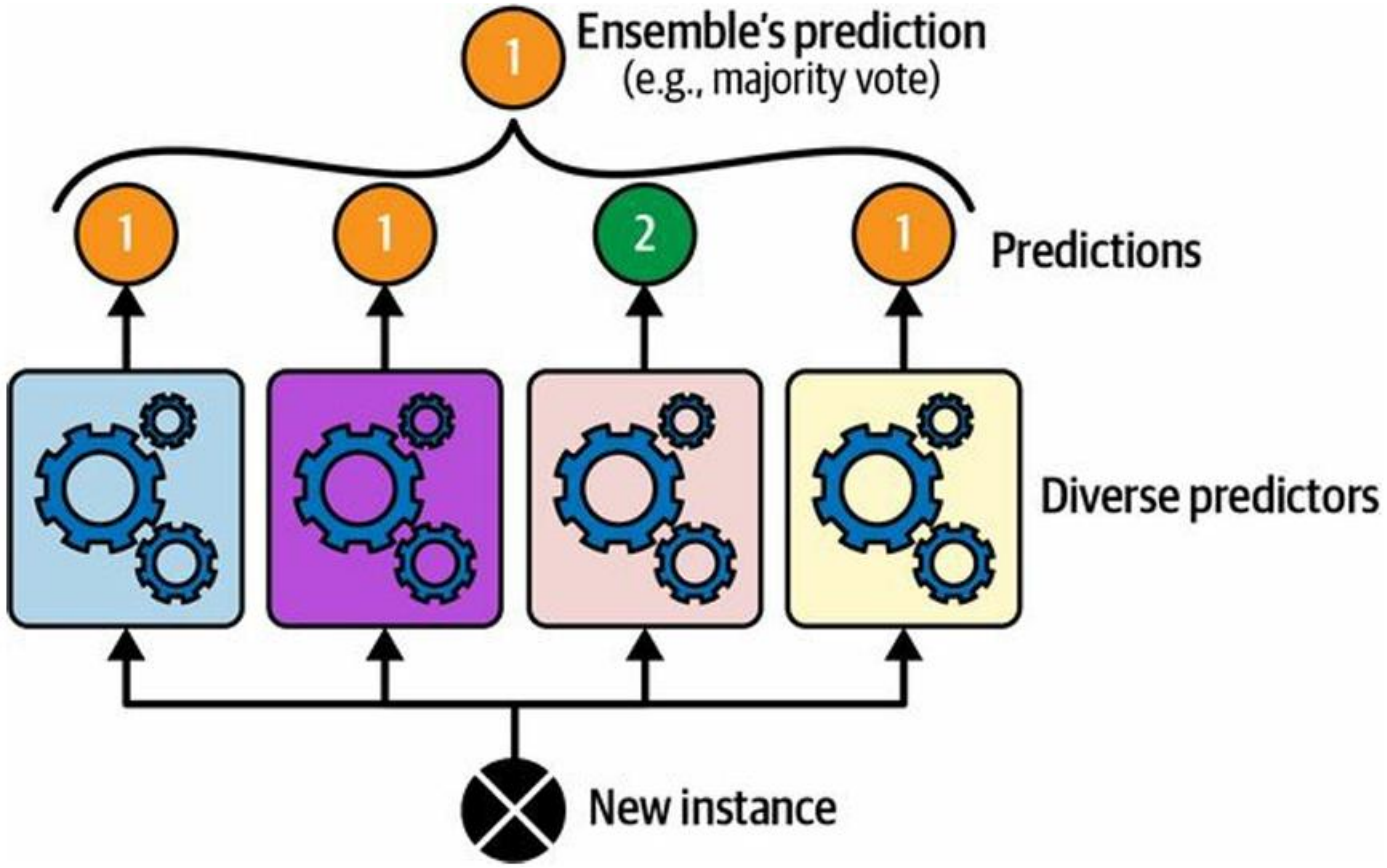
فرض کنید چند طبقه‌بندِ مختلف را آموزش داده‌اید که هر کدام تقریباً ۸۰% دقت دارند. این مجموعه می‌تواند شامل یک طبقه‌بند رگرسیون لجستیک، یک SVM، یک جنگل تصادفی، یک طبقه‌بند k-نزدیک‌ترین همسایه و شاید چند مدل دیگر باشد (به شکل ۷-۱ مراجعه کنید).



شکل ۷-۱. آموزش طبقه‌بندهای متنوع

یک روش بسیار ساده برای ایجاد یک طبقه‌بند قوی‌تر، تجمیع پیش‌بینی‌هایِ تمامِ طبقه‌بندها است: کلاسی که بیشترین رأی را کسب کند، پیش‌بینی نهایی گروه خواهد بود. این نوع طبقه‌بند را «طبقه‌بند رأی اکثریت یا رأی‌گیری سخت (*hard voting*)» می‌نامند.–این روش یکی از ساده‌ترین و رایج‌ترین متدهای Ensemble است که با استفاده از اصل اکثریت آرا خروجی نهایی را مشخص می‌کند(به شکل ۷-۲ مراجعه کنید).

© 2025 Mohsen Zare. All rights reserved.



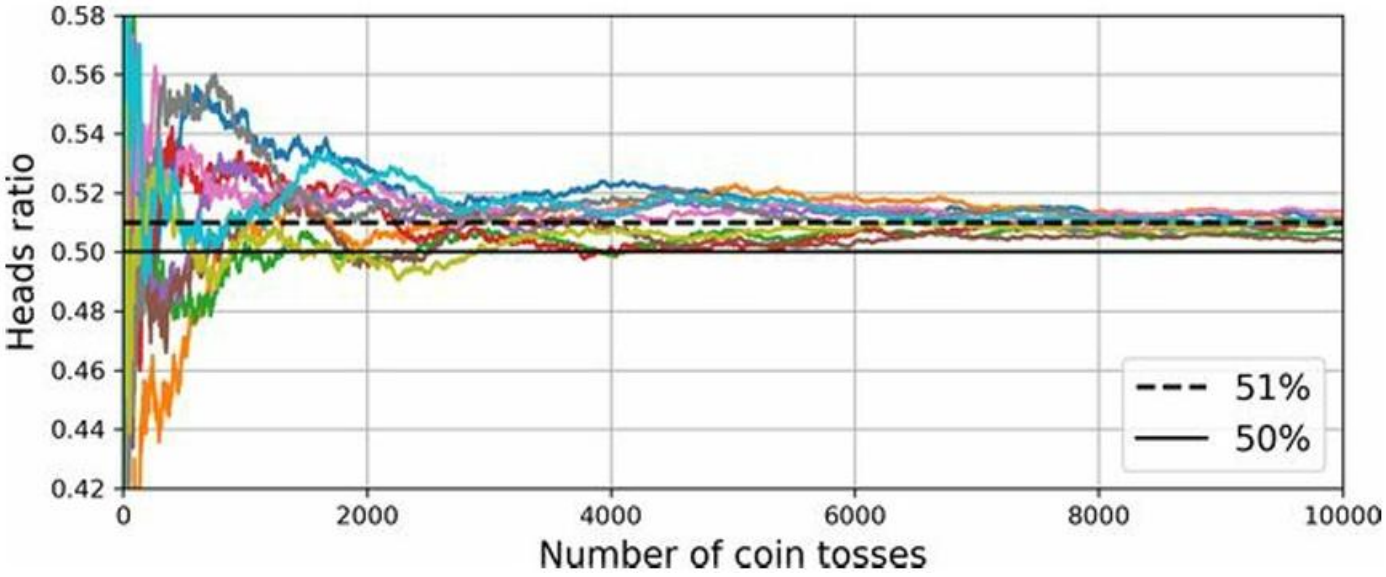
شکل ۷-۲. پیش‌بینی‌های طبقه‌بند رأی‌گیری سخت

تا حدودی شگفت‌آور است که این طبقه‌بند رأی‌گیری اغلب دقتی بالاتر از بهترین طبقه‌بند منفرد در «مجموعه(ensemble)» به دست می‌آورد. در واقع، حتی اگر هر طبقه‌بندی‌کننده یک «مدل یادگیرنده ضعیف (*weak learner*)» باشد، یعنی تنها کمی بهتر از حدس تصادفی عمل کند، کل مجموعه می‌تواند به یک «مدل یادگیرنده قوی (*strong learner*)» تبدیل شود و دقت بالایی کسب کند؛ البته به شرط آنکه تعداد کافی از مدل‌های یادگیرنده ضعیف در مجموعه وجود داشته باشند و از تنوع کافی برخوردار باشند.

چطور چنین چیزی ممکن است؟ یک تشبیه ساده می‌تواند این موضوع را روشن کند. فرض کنید یک سکه کمی بایاس دارد؛ یعنی ۵۱% احتمال دارد شیر و ۴۹% احتمال دارد خط بیاید. اگر این سکه را ۱,۰۰۰ بار پرتاب کنید، معمولاً حدود ۵۱۰ شیر و ۴۹۰ خط به دست می‌آید و بنابراین اکثریت شیرها خواهد بود. با محاسبه، متوجه می‌شوید که احتمال اکثریت آمدن شیر پس از ۱,۰۰۰ پرتاب تقریباً ۷۵% است. هرچه تعداد پرتاب‌ها افزایش یابد، این احتمال بیشتر می‌شود؛ برای مثال، با ۱۰,۰۰۰ پرتاب، احتمال اکثریت شیرها به بیش از ۹۷% می‌رسد. این پدیده ناشی از «قانون اعداد بزرگ (*law of large numbers*)»

© 2025 Mohsen Zare. All rights reserved.

است: هرچه تعداد پرتاب‌ها بیشتر شود، نسبت شیرها به احتمال واقعی شیر آمدن (۵۱%) نزدیک و نزدیک‌تر می‌شود. شکل ۷-۳، ده سِری پرتاب سکه با بایاس مشخص را نشان می‌دهد. همان‌طور که مشاهده می‌کنید، با افزایش تعداد پرتاب‌ها، نسبت شیرها به ۵۱% نزدیک می‌شود و در نهایت، تمام سری‌ها به طور مداوم بالاتر از ۵۰% قرار می‌گیرند.



شکل ۷-۳. قانون اعداد بزرگ

به همین ترتیب، تصور کنید یک گروه ۱,۰۰۰ طبقه‌بند داشته باشید که هرکدام به تنهایی فقط ۵۱% دقت دارند؛ یعنی تنها کمی بهتر از حدس تصادفی عمل می‌کنند. اگر بتوانید پیش‌بینی نهایی را بر اساس کلاسی که بیشترین رأی را کسب کرده است انجام دهید، می‌توانید به دقت حدود ۷۵% امیدوار باشید! با این حال، این ایده تنها زمانی صادق است که تمام طبقه‌بندها کاملاً مستقل باشند و خطاهایشان غیرهمبسته باشد. واضح است که در عمل چنین نیست، زیرا همه آنها روی داده‌های یکسان آموزش دیده‌اند و احتمالاً خطاهای مشابهی مرتکب می‌شوند. در نتیجه، ممکن است اکثریت آرا به سمت کلاس اشتباه گرایش پیدا کند و دقت کلی گروه کاهش یابد.

نکته

روش‌های ترکیبی(تجمیعی) بیشترین عملکرد را زمانی نشان می‌دهند که پیش‌بین‌ها تا حد ممکن مستقل از یکدیگر باشند. یکی از راه‌ها برای ایجاد طبقه‌بندهای متنوع، آموزش آنها با استفاده از الگوریتم‌های کاملاً متفاوت است. این تنوع باعث می‌شود که هر مدل انواع متفاوتی از خطاها را مرتکب شود و در نتیجه دقت کلی گروه افزایش یابد.

© 2025 Mohsen Zare. All rights reserved.

کتابخانه Scikit-Learn کلاس VotingClassifier را ارائه می‌دهد که کار با آن بسیار ساده است: کافی است لیستی از جفت‌های نام و پیش‌بین را به آن بدهید و سپس همانند یک طبقه‌بند معمولی از آن استفاده کنید. بیایید این کلاس را روی مجموعه داده‌های moons (که در فصل ۵ معرفی شد) آزمایش کنیم. ابتدا داده‌ها را بارگذاری و به دو بخش آموزشی و آزمون تقسیم می‌کنیم. سپس یک طبقه‌بند رأی‌گیری شامل سه طبقه‌بند متنوع ایجاد کرده و آموزش می‌دهیم:

```
from sklearn.datasets import make_moons
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

voting_clf = VotingClassifier(
  estimators=[
    ('lr', LogisticRegression(random_state=42)),
    ('rf', RandomForestClassifier(random_state=42)),
    ('svc', SVC(random_state=42))
  ]
)
voting_clf.fit(X_train, y_train)
```

هنگامی که یک VotingClassifier را برازش می‌دهید، این تابع هر پیش‌بین را شبیه‌سازی یا کلون (clone) می‌کند و سپس کلون‌ها را برازش می‌دهد.

- تخمین‌گرهای اصلی از طریق ویژگی estimators در دسترس هستند.
- کلون‌های برازش شده از طریق ویژگی estimators_ قابل دسترسی‌اند.

اگر ترجیح می‌دهید به جای لیست از یک دیکشنری (dict) استفاده کنید، می‌توانید از ویژگی‌های named_estimators یا named_estimators_ بهره ببرید. برای شروع، بیایید دقت هر پیش‌بین برازش‌شده را روی مجموعه آزمون بررسی کنیم:

© 2025 Mohsen Zare. All rights reserved.

```
>>> for name, clf in voting_clf.named_estimators_.items():
…   print(name, "=", clf.score(X_test, y_test))
…
lr = 0.864
rf = 0.896
svc = 0.896
```

هنگامی که متد predict() مربوط به طبقه‌بند رأی‌گیری فراخوانی می‌شود، فرآیند رأی‌گیری سخت اجرا می‌گردد. برای مثال، اگر در اولین نمونه از مجموعه‌ی آزمون، دو تا از سه پیش‌بینی‌کننده کلاس ۱ را پیش‌بینی کنند، طبقه‌بند رأی‌گیری نیز همین کلاس را به عنوان پیش‌بینی نهایی انتخاب می‌کند:

```
>>> voting_clf.predict(X_test[:1])
array([1])
>>> [clf.predict(X_test[:1]) for clf in voting_clf.estimators_]
[array([1]), array([1]), array([0])]
```

حالا بیایید عملکرد طبقه‌بند رأی‌گیری را روی مجموعه‌ی آزمون بررسی کنیم:

```
>>> voting_clf.score(X_test, y_test)
0.912
```

طبقه‌بندی رأی‌گیری معمولاً از تمام طبقه‌بندهای منفرد عملکرد بهتری دارد. اگر همه‌ی طبقه‌بندها قادر باشند احتمال تعلق هر نمونه به کلاس‌ها را تخمین بزنند (یعنی متد predict_proba() را داشته باشند)، می‌توانید به Scikit-Learn دستور دهید کلاسی را پیش‌بینی کند که میانگین بالاترین احتمال را در میان تمام طبقه‌بندها دارد. این رویکرد، «رأی‌گیری نرم (*soft voting*)» نامیده می‌شود. در بسیاری از موارد، رأی‌گیری نرم دقت بیشتری نسبت به رأی‌گیری سخت دارد، زیرا پیش‌بینی‌هایی که با اطمینان بالاتر انجام می‌شوند، وزن بیشتری پیدا می‌کنند. برای فعال کردن این حالت، کافی است هایپرپارامتر voting را روی "soft" تنظیم کنید و اطمینان حاصل نمایید که همه‌ی طبقه‌بندها قادر به تخمین احتمالات کلاس هستند. نکته: این قابلیت به طور پیش‌فرض برای کلاس SVC فعال نیست. بنابراین باید پارامتر probability=True را تنظیم کنید. با این کار، SVC از اعتبارسنجی متقاطع برای تخمین احتمال کلاس‌ها استفاده می‌کند، فرآیند آموزش کمی کندتر می‌شود، اما در عوض متد predict_proba() نیز در دسترس خواهد بود.

© 2025 Mohsen Zare. All rights reserved.

بیایید آن را امتحان کنیم:

```
>>> voting_clf.voting = "soft"
>>> voting_clf.named_estimators["svc"].probability = True
>>> voting_clf.fit(X_train, y_train)
>>> voting_clf.score(X_test, y_test)
0.92
```

تنها با به‌کارگیری رأی‌گیری نرم توانستیم به دقت ۹۲% دست پیدا کنیم – نتیجه‌ی قابل‌توجهی است!

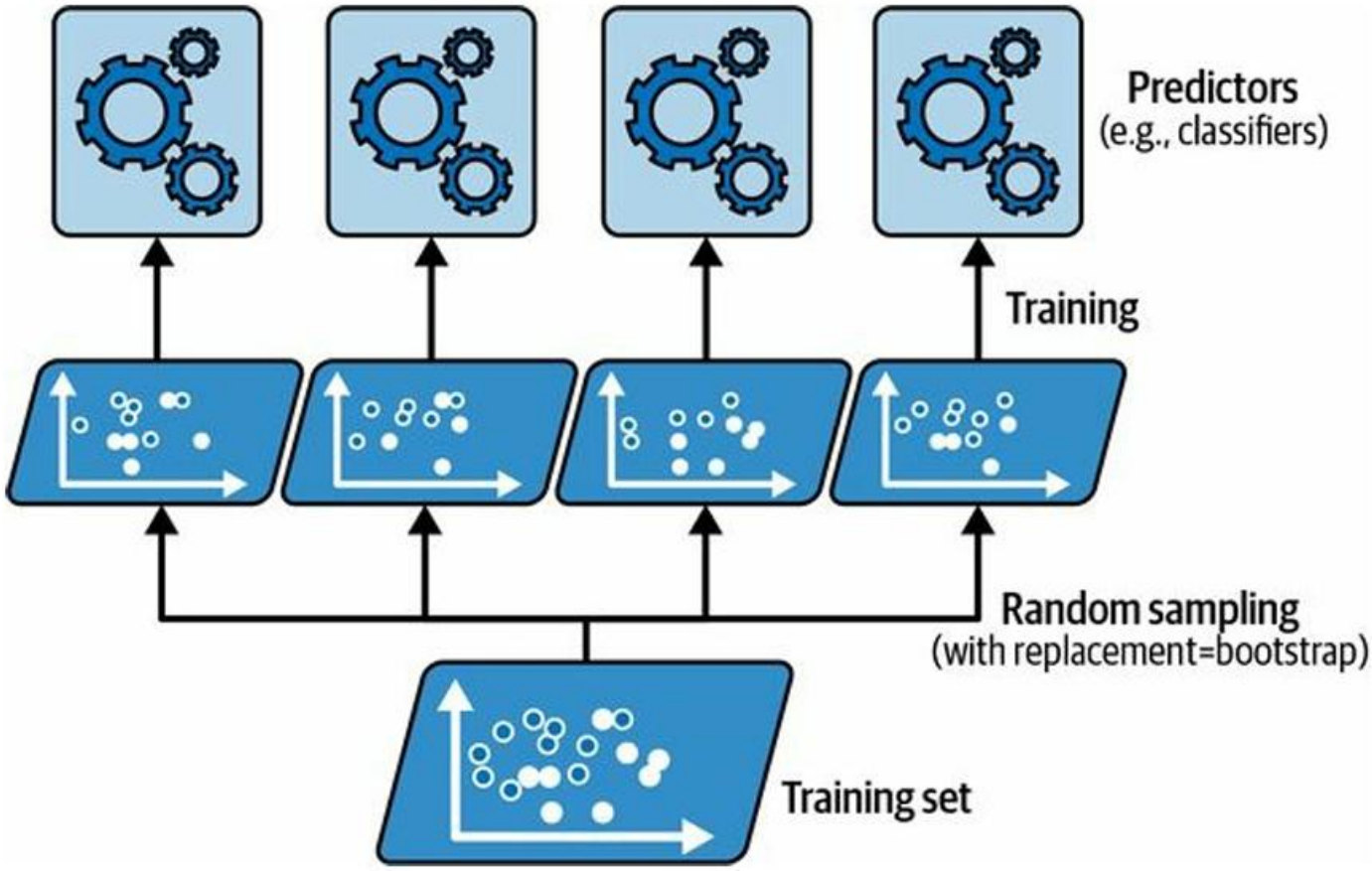
© 2025 Mohsen Zare. All rights reserved.

Bagging و Pasting

یکی از راه‌های ایجاد مجموعه‌ای متنوع از طبقه‌بندها، همان‌طور که پیش‌تر اشاره شد، استفاده از الگوریتم‌های آموزشی متفاوت است. اما یک رویکرد دیگر این است که از یک الگوریتم آموزشی ثابت استفاده کنیم، با این تفاوت که آن را روی زیرمجموعه‌های تصادفی مختلفی از داده‌های آموزشی اجرا کنیم. اگر این نمونه‌گیری به‌صورت «با جایگزینی» انجام شود، روش به‌دست‌آمده را «بَگینگ (*bagging*)» می‌نامند (که کوتاه‌شده‌ی (*bootstrap aggregating*) به معنی تجمیع بوت‌اِستِرَپ یا نمونه‌گیری بازگردانی است). در مقابل، اگر نمونه‌گیری به‌صورت «بدون جایگزینی» صورت گیرد، این روش «پِیستینگ (*pasting*)»–نمونه‌گیری بدون بازگردانی؛ داده‌ها را بدون بازگردانی برای ساخت زیرمجموعه‌های آموزشی انتخاب می‌کند–نامیده می‌شود.

به بیان دیگر، در هر دو روش بَگینگ و چسباندن، هر نمونه‌ی آموزشی می‌تواند در مجموعه‌های آموزشی مختلفی که برای پیش‌بین‌ها ساخته می‌شوند حضور داشته باشد، اما تنها در بَگینگ این امکان وجود دارد که یک نمونه‌ی آموزشی چندین بار در مجموعه‌ی آموزشی یک پیش‌بین تکرار شود. روند این نمونه‌گیری و آموزش در شکل ۷-۴ نمایش داده شده است.

© 2025 Mohsen Zare. All rights reserved.



شکل ۷-۴. بَگینگ و پِیستینگ شامل آموزش چندین پیش‌بین روی نمونه‌های تصادفی مختلف از مجموعه آموزشی است

پس از آنکه همه‌ی پیش‌بینی‌کننده‌ها آموزش دیدند، می‌توان با ترکیب خروجی آن‌ها یک پیش‌بینی نهایی برای نمونه‌های جدید به دست آورد. روش تجمیعی معمولاً به این صورت است که در مسائل طبقه‌بندی از « نَما (مُد) آماری (*statistical mode*)» استفاده می‌شود (یعنی پرتکرارترین برچسب، مشابه رأی‌گیری سخت؛ شاخص مرکزی داده‌ها که نشان‌دهنده مقداری است که بیشترین فراوانی را دارد، معمولاً برای داده‌های طبقه‌ای یا گسسته بیشتر استفاده می‌شود)، و در مسائل رگرسیون از میانگین خروجی‌ها. هر پیش‌بین منفرد، نسبت به حالتی که روی کل مجموعه‌ی آموزشی اصلی آموزش داده شود، بایاس بیشتری خواهد داشت. اما با ترکیب آن‌ها، هم بایاس و هم واریانس کاهش می‌یابد. در مجموع، نتیجه این است که گروه نهایی دارای بایاسی مشابه ولی واریانسی کمتر از یک پیش‌بین منفرد آموزش‌دیده روی کل داده‌ها خواهد بود.

همان‌طور که در شکل ۷-۴ نشان داده شده است، می‌توان پیش‌بین‌ها را به‌صورت موازی، روی هسته‌های مختلف CPU یا حتی روی چند سرور مجزا آموزش داد. همچنین، فرایند پیش‌بینی نیز

© 2025 Mohsen Zare. All rights reserved.

می‌تواند به‌صورت موازی انجام شود. این ویژگی باعث شده است که روش‌های bagging و pasting بسیار محبوب باشند، زیرا از مقیاس‌پذیری بالایی برخوردارند.

© 2025 Mohsen Zare. All rights reserved.

بگینگ و پِیستینگ در Scikit-Learn

کتابخانه‌ی Scikit-Learn یک API ساده برای پیاده‌سازی Bagging و Pasting فراهم می‌کند: کلاس BaggingClassifier (و برای مسائل رگرسیون، کلاس BaggingRegressor). در کد زیر، یک گروه شامل ۵۰۰ طبقه‌بندی درخت تصمیم ایجاد و آموزش داده می‌شود. هر درخت روی ۱۰۰ نمونه‌ی آموزشی که به‌صورت تصادفی و با جایگزینی از مجموعه‌ی آموزشی انتخاب شده‌اند، آموزش می‌بیند (این حالت نمونه‌ای از بَگینگ است). اگر بخواهید به‌جای آن از پِیستینگ استفاده کنید، کافی است پارامتر bootstrap=False را تنظیم نمایید. پارامتر n_jobs مشخص می‌کند که Scikit-Learn از چند هسته‌ی CPU برای آموزش و پیش‌بینی استفاده کند. اگر مقدار آن روی ۱- قرار گیرد، همه‌ی هسته‌های موجود به‌کار گرفته می‌شوند:

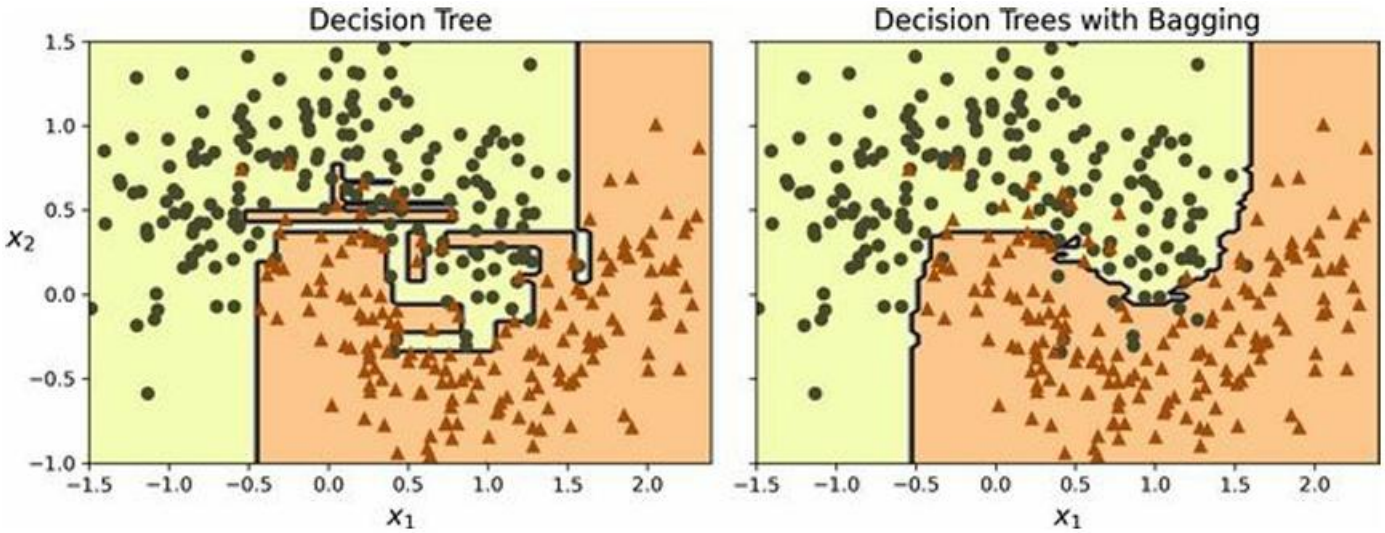
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(DecisionTreeClassifier(), n_estimators=500,
            max_samples=100, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
```

> **نکته**
>
> اگر طبقه‌بند پایه قادر به تخمین احتمالات کلاس باشد (یعنی متد predict_proba()) را پیاده‌سازی کند)، کلاس BaggingClassifier به‌صورت خودکار از رأی‌گیری نرم به جای رأی‌گیری سخت استفاده می‌کند. این قابلیت به‌ویژه برای درخت‌های تصمیم‌گیری نیز صادق است.

شکل ۷-۵ مرز تصمیم یک درخت تصمیم واحد را با مرز تصمیم یک گروه بَگینگ شامل ۵۰۰ درخت (همان‌طور که در کد قبلی آموزش داده شد) مقایسه می‌کند. همان‌طور که مشاهده می‌کنید، پیش‌بینی‌های این گروه احتمالاً تعمیم‌پذیری بهتری نسبت به یک درخت تنها خواهند داشت: گروه دارای بایاس مشابه اما واریانس کمتر است؛ یعنی تعداد خطاها روی مجموعه آموزش تقریباً یکسان است، اما مرز تصمیم‌گیری به شکل قابل توجهی منظم‌تر و صاف‌تر است.

© 2025 Mohsen Zare. All rights reserved.

بَگینگ با ایجاد تنوع بیشتر در زیرمجموعه‌هایی که هر پیش‌بینی‌کننده روی آن‌ها آموزش می‌بیند، معمولاً با بایاس کمی بالاتر نسبت به پِیستینگ همراه است؛ اما این تنوع اضافی باعث می‌شود که پیش‌بین‌ها همبستگی کمتری داشته باشند و در نتیجه واریانس گروه کاهش یابد. در نتیجه، بَگینگ اغلب به مدل‌های قوی‌تر و قابل اعتمادتر منجر می‌شود، که توضیح می‌دهد چرا معمولاً ترجیح داده می‌شود. با این حال، اگر زمان و قدرت پردازشی کافی دارید، می‌توانید با استفاده از اعتبارسنجی متقاطع هر دو روش (bagging و pasting) را ارزیابی کرده و روشی را که بهترین عملکرد را دارد، انتخاب کنید.



شکل ۷-۵. یک درخت تصمیم واحد (چپ) در مقابل یک گروه بگینگ متشکل از ۵۰۰ درخت (راست)

© 2025 Mohsen Zare. All rights reserved.

اززیابی داده‌های خارج از کیسه(OOB)

در روش بگینگ، هنگام نمونه‌گیری با جایگزینی برای هر پیش‌بین، برخی از نمونه‌های آموزشی ممکن است چندین بار انتخاب شوند و برخی دیگر اصلاً انتخاب نشوند. به طور پیش‌فرض، یک BaggingClassifier از m نمونه آموزشی با جایگزینی (bootstrap=True) استفاده می‌کند، که m اندازه مجموعه آموزشی است. با این فرآیند، به طور ریاضی نشان داده شده است که به طور متوسط تنها حدود ۶۳% از نمونه‌ها برای هر پیش‌بینی‌کننده انتخاب می‌شوند و ۳۷% باقی‌مانده که انتخاب نشده‌اند، به عنوان «نمونه‌های خارج از کیسه (OOB: *out-of-bag*)» شناخته می‌شوند.

یک گروه بگینگ می‌تواند با استفاده از نمونه‌های OOB ارزیابی شود، بدون آنکه نیاز به مجموعه اعتبارسنجی جداگانه‌ای باشد. در واقع، اگر تعداد کافی پیش‌بینی‌کننده وجود داشته باشد، هر نمونه آموزشی احتمالاً در چندین پیش‌بینی‌کننده به عنوان نمونه OOB ظاهر می‌شود. بنابراین می‌توان از پیش‌بینی‌های این پیش‌بینی‌کننده‌ها برای محاسبه یک پیش‌بینی گروهی منصفانه برای آن نمونه استفاده کرد. پس از داشتن پیش‌بینی برای تمام نمونه‌ها، می‌توان دقت گروه یا هر معیار ارزیابی دیگری را محاسبه نمود.

در Scikit-Learn، می‌توانید با تنظیم oob_score=True هنگام ایجاد یک BaggingClassifier، درخواست ارزیابی خودکار OOB پس از آموزش را بدهید. نمونه کد زیر این روند را نشان می‌دهد. امتیاز حاصل از این ارزیابی در ویژگی oob_score_ ذخیره می‌شود:

```
>>> bag_clf = BaggingClassifier(DecisionTreeClassifier(), n_estimators=500,
…                            oob_score=True, n_jobs=-1, random_state=42)
…
>>> bag_clf.fit(X_train, y_train)
>>> bag_clf.oob_score_
0.896
```

بر اساس این ارزیابی OOB، انتظار می‌رود این BaggingClassifier روی مجموعه آزمون دقت تقریبی ۸۹٫۶٪ داشته باشد. بیایید این موضوع را بررسی کنیم:

```
>>> from sklearn.metrics import accuracy_score
>>> y_pred = bag_clf.predict(X_test)
```

© 2025 Mohsen Zare. All rights reserved.

```
>>> accuracy_score(y_test, y_pred)
0.92
```

در این آزمون، به دقت ۹۲% رسیدیم؛ بنابراین ارزیابی OOB کمی بدبینانه بود و حدود ۲% کمتر از دقت واقعی نشان داد.

همچنین، تابع تصمیم‌گیری OOB برای هر نمونه آموزشی از طریق ویژگی oob_decision_function_ قابل دسترسی است. از آنجا که تخمین‌گر پایه دارای متد predict_proba() است، این تابع احتمال تعلق هر نمونه به کلاس‌ها را برمی‌گرداند. برای مثال، ارزیابی OOB نشان می‌دهد که اولین نمونه آموزشی با احتمال ۶۷٫۶ % به کلاس مثبت و با احتمال ۳۲٫۴ % به کلاس منفی تعلق دارد:

```
>>> bag_clf.oob_decision_function_[:3] # probas for the first 3 instances
array([[0.32352941, 0.67647059],
       [0.3375  , 0.6625  ],
       [1.      , 0.      ]])
```

© 2025 Mohsen Zare. All rights reserved.

تکه‌های تصادفی و زیرفضاهای تصادفی

کلاس BaggingClassifier امکان نمونه‌گیری تصادفی از ویژگی‌ها را نیز فراهم می‌کند. این نمونه‌گیری توسط پارامترهای max_features و bootstrap_features کنترل می‌شود. عملکرد این پارامترها مشابه max_samples و bootstrap است، با این تفاوت که به جای نمونه‌ها، روی ویژگی‌ها اعمال می‌شوند. در نتیجه، هر پیش‌بین روی یک زیرمجموعه تصادفی از ویژگی‌های ورودی آموزش داده خواهد شد.

این تکنیک مخصوصاً زمانی مفید است که با داده‌های با ابعاد بالا، مانند تصاویر، سر و کار دارید، زیرا می‌تواند سرعت آموزش را به طور قابل توجهی افزایش دهد.

- زمانی که هم از نمونه‌ها و هم از ویژگی‌ها نمونه‌گیری می‌کنیم، به این روش «تکه‌های تصادفی (*random patches method*)» گفته می‌شود.
- اگر همه نمونه‌های آموزشی حفظ شوند (bootstrap=False و max_samples=1.0) ولی از ویژگی‌ها نمونه‌گیری شود (bootstrap_features=True یا max_features کمتر از ۱/۰)، این روش به نام «زیرفضاهای تصادفی (*random subspaces method*)» شناخته می‌شود.

نمونه‌گیری ویژگی‌ها باعث افزایش تنوع پیش‌بین‌ها می‌شود و در مقابل، بایاس کمی بالاتر را به همراه کاهش واریانس به ارمغان می‌آورد.

© 2025 Mohsen Zare. All rights reserved.

جنگل‌های تصادفی

همان‌طور که پیش‌تر گفتیم، یک جنگل تصادفی مجموعه‌ای از درخت‌های تصمیم‌گیری است که معمولاً با روش بَگینگ (و گاهی پِیستینگ) آموزش داده می‌شوند و اغلب از کل مجموعه آموزشی (با max_samples برابر اندازه مجموعه) استفاده می‌کنند. به جای ساخت یک BaggingClassifier و ارسال DecisionTreeClassifier به آن، می‌توان از کلاس RandomForestClassifier بهره برد که برای درخت‌های تصمیم بهینه‌تر و راحت‌تر است. به‌طور مشابه، کلاس RandomForestRegressor برای مسائل رگرسیون وجود دارد. کد زیر یک طبقه‌بندی‌کننده جنگل تصادفی شامل ۵۰۰ درخت، هر کدام با حداکثر ۱۶ گره برگ، را با استفاده از تمام هسته‌های CPU موجود آموزش می‌دهد:

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16,
            n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

با چند استثنا، یک RandomForestClassifier تمامی هایپرپارامترهای DecisionTreeClassifier را برای کنترل رشد درخت‌ها در اختیار دارد، به‌علاوه تمام هایپرپارامترهای BaggingClassifier که برای مدیریت رفتار گروه استفاده می‌شوند.

الگوریتم جنگل تصادفی هنگام رشد درختان، تصادفی‌سازی بیشتری اعمال می‌کند؛ به جای جستجوی بهترین ویژگی در میان تمام ویژگی‌ها برای تقسیم هر گره (همان‌طور که در فصل ۶ دیدیم)، بهترین ویژگی را تنها در میان یک زیرمجموعه تصادفی از ویژگی‌ها انتخاب می‌کند. به طور پیش‌فرض، این زیرمجموعه شامل n ویژگی است (که n تعداد کل ویژگی‌ها است). این کار باعث تنوع بیشتر درخت‌ها می‌شود، که دوباره بایاس بالاتر را با واریانس پایین‌تر تعادل می‌بخشد و معمولاً منجر به مدل بهتری

© 2025 Mohsen Zare. All rights reserved.

می‌گردد. بنابراین، یک BaggingClassifier مشابه، معادل RandomForestClassifier قبلی خواهد بود:

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(max_features="sqrt", max_leaf_nodes=16),
    n_estimators=500, n_jobs=-1, random_state=42)
```

© 2025 Mohsen Zare. All rights reserved.

درختانِ با تصادفی‌سازی شدید

هنگام رشد یک درخت در جنگل تصادفی، همان‌طور که پیش‌تر گفتیم، در هر گره تنها یک زیرمجموعه تصادفی از ویژگی‌ها برای تقسیم بررسی می‌شود. می‌توان درجه تصادفی‌سازی را با استفاده از آستانه‌های تصادفی برای هر ویژگی افزایش داد؛ به این صورت که به جای جستجوی بهترین آستانه ممکن (مانند درخت‌های تصمیم استاندارد)، تقسیم‌ها به طور تصادفی انتخاب شوند. برای فعال کردن این حالت، کافی است هنگام ساخت DecisionTreeClassifier، پارامتر "splitter="random را تنظیم کنید.

جنگلی متشکل از چنین درختان به شدت تصادفی، گروهی از «درختان به شدت تصادفی (*extremely randomized trees*) که به اختصار *Extra-Trees*» نامیده می‌شود. این تکنیک نیز بایاس بالاتر را با واریانس کمتر معاوضه می‌کند. علاوه بر این، آموزش طبقه‌بندهای Extra-Trees نسبت به جنگل‌های تصادفی معمولی بسیار سریع‌تر است، زیرا نیازی به یافتن بهترین آستانه برای هر ویژگی در هر گره ندارد که یکی از وقت‌گیرترین مراحل رشد درخت است.

می‌توانید با استفاده از کلاس ExtraTreesClassifier در Scikit-Learn، یک طبقه‌بند Extra-Trees بسازید. رابط کاربری (API) آن مشابه RandomForestClassifier است، با این تفاوت که مقدار پیش‌فرض bootstrap برابر با False است. به همین ترتیب، کلاس ExtraTreesRegressor همان API کلاس RandomForestRegressor را دارد، اما مقدار پیش‌فرض bootstrap روی False تنظیم شده است.

نکته

نمی‌توان از قبل گفت که آیا یک RandomForestClassifier عملکرد بهتری نسبت به ExtraTreesClassifier خواهد داشت یا خیر. به طور کلی، تنها راه مطمئن برای فهمیدن این موضوع، آموزش هر دو مدل و مقایسه عملکردشان با استفاده از اعتبارسنجی متقاطع است.

© 2025 Mohsen Zare. All rights reserved.

اهمیت ویژگی

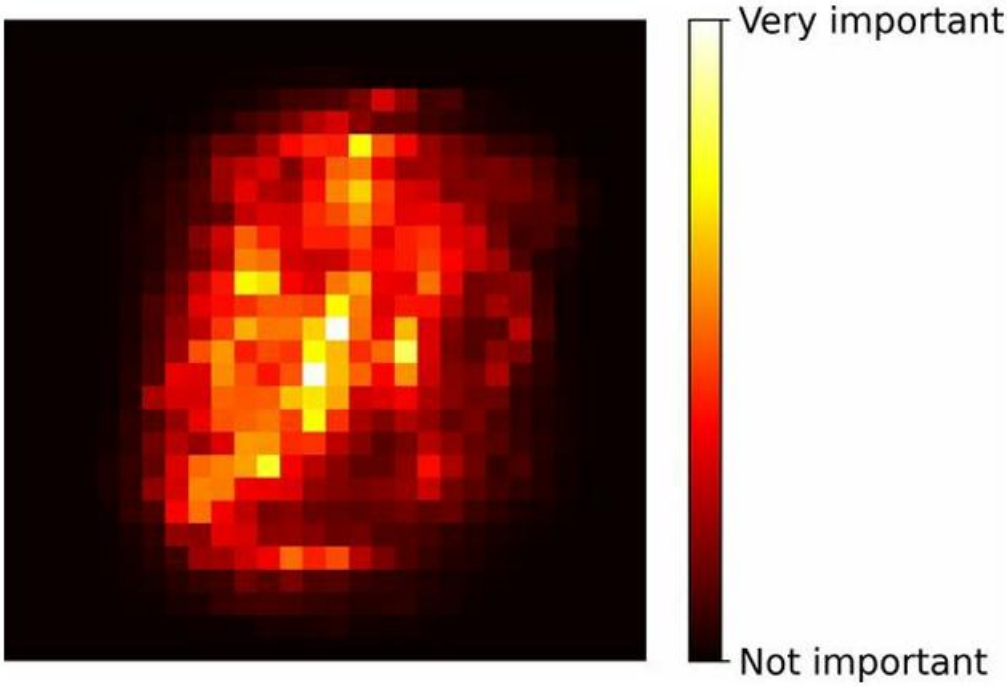
یکی دیگر از قابلیت‌های برجسته‌ی جنگل‌های تصادفی، توانایی ساده در تعیین اهمیت نسبی هر ویژگی است. در Scikit-Learn، اهمیت یک ویژگی با سنجش میزان کاهش ناخالصی در گره‌هایی که از آن ویژگی استفاده کرده‌اند، در تمام درختان جنگل محاسبه می‌شود. دقیق‌تر بگوییم، این یک میانگین وزنی است که وزن هر گره با تعداد نمونه‌های آموزشی مرتبط با آن تعیین می‌شود (رجوع شود به فصل ۶).

Scikit-Learn این امتیاز را پس از آموزش به صورت خودکار برای هر ویژگی محاسبه کرده و سپس نتایج را طوری مقیاس‌بندی می‌کند که مجموع اهمیت همه ویژگی‌ها برابر با ۱ شود. می‌توان با استفاده از متغیر feature_importances_ به این مقادیر دسترسی داشت. برای مثال، کد زیر یک RandomForestClassifier را روی مجموعه داده‌ی iris (معرفی شده در فصل ۴) آموزش می‌دهد و اهمیت هر ویژگی را نمایش می‌دهد. به نظر می‌رسد که طول گلبرگ (۴۴%) و عرض گلبرگ (۴۲%) مهم‌ترین ویژگی‌ها هستند، در حالی که طول و عرض کاسبرگ نسبتاً کم‌اهمیت‌اند (به ترتیب ۱۱% و ۲%):

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris(as_frame=True)
>>> rnd_clf = RandomForestClassifier(n_estimators=500, random_state=42)
>>> rnd_clf.fit(iris.data, iris.target)
>>> for score, name in zip(rnd_clf.feature_importances_, iris.data.columns):
...     print(round(score, 2), name)
...
0.11 sepal length (cm)
0.02 sepal width (cm)
0.44 petal length (cm)
0.42 petal width (cm)
```

به همین ترتیب، اگر یک طبقه‌بندی جنگل تصادفی را روی مجموعه داده‌ی MNIST (معرفی شده در فصل ۳) آموزش دهید و اهمیت هر پیکسل را ترسیم کنید، تصویری مشابه شکل ۷-۶ به دست خواهید آورد.

© 2025 Mohsen Zare. All rights reserved.



شکل ۷-۶. اهمیت پیکسل MNIST (طبق یک طبقه‌بند جنگل تصادفی)

جنگل‌های تصادفی برای فهم سریع اهمیت واقعی ویژگی‌ها بسیار مفید هستند، به‌ویژه وقتی که قصد انجام انتخاب ویژگی (feature selection) را دارید.

© 2025 Mohsen Zare. All rights reserved.

تقویت

«تقویت (Boosting)»، که در ابتدا با نام «تقویت فرضیه (hypothesis boosting)» شناخته می‌شد، به هر روش ترکیبی ای اطلاق می‌شود که چندین مدل یادگیرنده‌ی ضعیف را به یک مدل یادگیرنده‌ی قوی ترکیب می‌کند. ایده اصلی اکثر روش‌های تقویتی این است که پیش‌بین‌ها به‌صورت متوالی آموزش داده شوند، به‌طوری که هر یک تلاش کند خطاهای مدل قبلی را اصلاح کند. روش‌های متعددی برای تقویت وجود دارد، اما تا کنون محبوب‌ترین آن‌ها «*AdaBoost*(مخفف تقویت تطبیقی یا adaptive boosting)» و «تقویت گرادیان (gradient boosting)» بوده‌اند. ابتدا با AdaBoost شروع می‌کنیم.

© 2025 Mohsen Zare. All rights reserved.

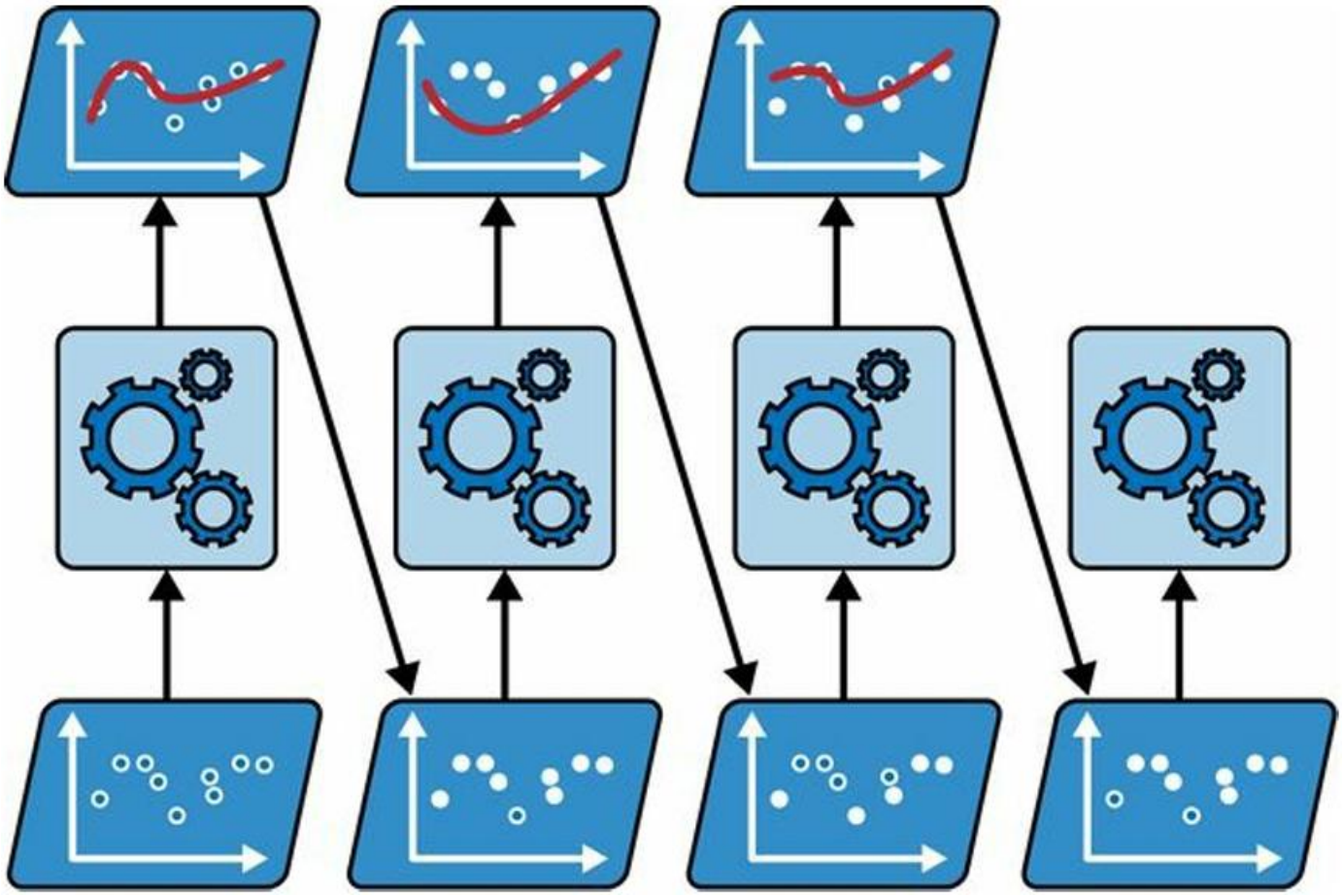
تقویت تطبیقی (AdaBoost)

یکی از روش‌ها برای اینکه یک پیش‌بین جدید بتواند خطاهای پیش‌بین قبلی را اصلاح کند، تمرکز بیشتر بر نمونه‌های آموزشی است که پیش‌بین قبلی روی آن‌ها عملکرد ضعیفی داشته است. این روند باعث می‌شود پیش‌بینی‌کنندگان بعدی توجه ویژه‌ای به نمونه‌های دشوار نشان دهند. این دقیقاً همان تکنیکی است که AdaBoost به کار می‌گیرد.

برای مثال، هنگام آموزش یک طبقه‌بند AdaBoost، الگوریتم ابتدا یک طبقه‌بند پایه (مانند درخت تصمیم) را آموزش می‌دهد و از آن برای پیش‌بینی روی مجموعه آموزشی استفاده می‌کند. سپس وزن نمونه‌های آموزشی که اشتباه طبقه‌بندی شده‌اند افزایش می‌یابد. با استفاده از این وزن‌های به‌روزرسانی‌شده، یک طبقه‌بند دوم آموزش داده می‌شود و دوباره پیش‌بینی‌ها انجام می‌شود و وزن‌ها دوباره تنظیم می‌شوند. این روند به همین ترتیب ادامه پیدا می‌کند (شکل ۷-۷ را ببینید).

شکل ۷-۸ مرزهای تصمیم پنج پیش‌بینی‌کننده متوالی را روی مجموعه داده moons نشان می‌دهد (در این مثال، هر پیش‌بین یک طبقه‌بندِ SVM با تنظیمات قوی و کِرنِل RBF است). طبقه‌بند اول، بسیاری از نمونه‌ها را اشتباه تشخیص می‌دهد و در نتیجه وزن این نمونه‌ها افزایش می‌یابد. طبقه‌بند دوم سپس توجه بیشتری به این نمونه‌ها می‌کند و عملکرد بهتری نشان می‌دهد و همین روند برای پیش‌بین‌های بعدی ادامه پیدا می‌کند. نمودار سمت راست همان توالی پیش‌بین‌ها را نشان می‌دهد، با این تفاوت که نرخ یادگیری نصف شده است، یعنی افزایش وزن نمونه‌های اشتباه در هر تکرار کمتر است. همان‌طور که مشاهده می‌کنید، این روش یادگیری ترتیبی شباهت‌هایی با گرادیان نزولی دارد، با این تفاوت که AdaBoost به جای اصلاح پارامترهای یک پیش‌بین برای مینیمم کردن تابع هزینه، پیش‌بین‌های جدید را به مجموعه اضافه می‌کند و به تدریج مدل را بهبود می‌بخشد.

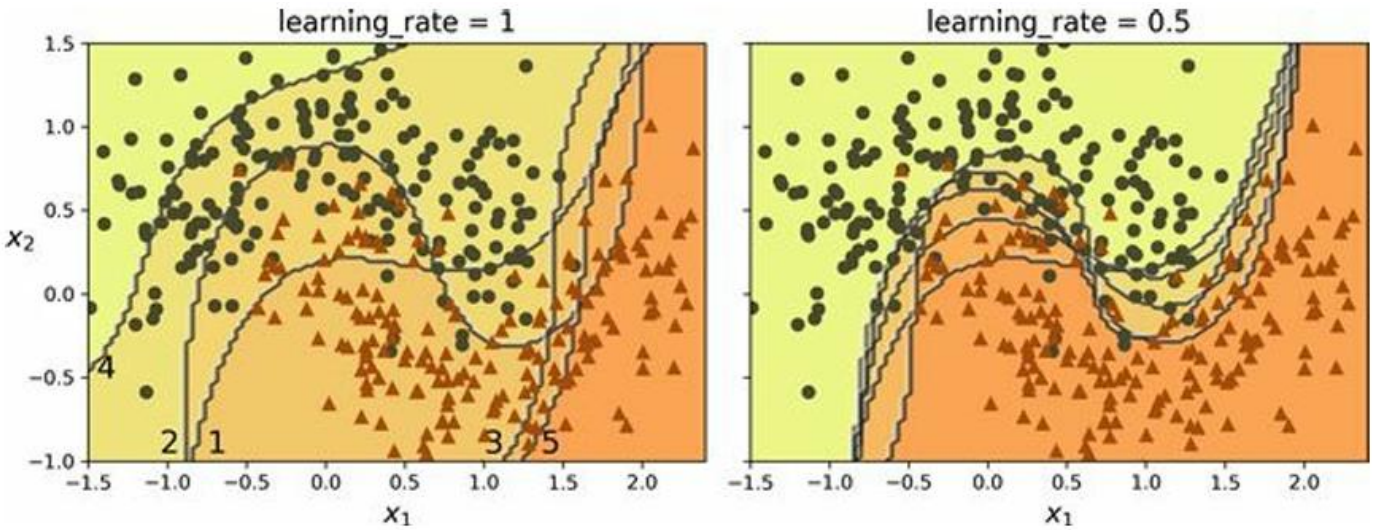
© 2025 Mohsen Zare. All rights reserved.



شکل ۷-۷. آموزشِ ترتیبی AdaBoost با به‌روزرسانی وزن نمونه‌ها

پس از آموزش تمام پیش‌بین‌ها، گروه پیش‌بینی‌هایی شبیه به روش‌های bagging یا pasting ارائه می‌دهد، با این تفاوت که هر پیش‌بین با توجه به دقت کلی‌اش روی مجموعه آموزشی، وزن متفاوتی در پیش‌بینی نهایی دارد.

© 2025 Mohsen Zare. All rights reserved.



شکل ۷-۸. مرزهای تصمیم‌گیریِ پیش‌بین‌هایِ متوالی

هشدار

یک محدودیت کلیدی در این روش یادگیری ترتیبی این است که آموزش پیش‌بین‌ها را نمی‌توان به‌صورت موازی انجام داد، زیرا هر پیش‌بین تنها پس از آموزش و ارزیابی پیش‌بین قبلی قابل آموزش است. بنابراین، برخلاف bagging یا pasting، این روش از نظر مقیاس‌پذیری محدودیت دارد.

بیایید الگوریتم AdaBoost را کمی دقیق‌تر بررسی کنیم. ابتدا، وزن هر نمونه $w^{(i)}$ روی $1/m$ قرار می‌گیرد. سپس یک پیش‌بینِ اولیه آموزش داده می‌شود و نرخ خطای وزنی آن، r_1، روی مجموعه‌ی آموزشی محاسبه می‌شود (به معادله ۷-۱ مراجعه کنید).

معادله ۷-۱. نرخ خطای وزنیِ پیش‌بینی‌کننده‌یِ j اُم

$$r_j = \sum_{i=1}^{m} w^{(i)} \, \mathbf{1}\left(\hat{y}_j^{(i)} \neq y^{(i)}\right)$$

که در آن $\hat{y}_j\,(i)$ پیش‌بینیِ پیش‌بینی‌کننده‌ی j اُم برای نمونه i اُم است.

© 2025 Mohsen Zare. All rights reserved.

سپس وزن پیش‌بینی‌کننده α_j با استفاده از معادله ۷-۲ محاسبه می‌شود، که در آن η هایپرپارامتر نرخِ یادگیری است و مقدار پیش‌فرض آن ۱ است. وزن پیش‌بین مستقیماً با دقت آن ارتباط دارد: هرچه پیش‌بین دقیق‌تر باشد، وزن آن بزرگ‌تر خواهد شد. اگر پیش‌بینی‌ها صرفاً حدسی و تصادفی باشند، وزن تقریباً صفر خواهد بود. در مقابل، اگر پیش‌بین بیشتر اوقات اشتباه کند—یعنی دقت آن کمتر از حدس تصادفی باشد—وزن آن منفی خواهد شد.

معادله ۷-۲. وزن پیش‌بینی‌کننده

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

سپس الگوریتم AdaBoost وزن هر نمونه را با استفاده از معادله ۷-۳ به‌روزرسانی می‌کند، به‌طوری که وزن نمونه‌هایی که اشتباه طبقه‌بندی شده‌اند افزایش می‌یابد.

معادله ۷-۳. قانون به‌روزرسانی وزن

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if} \quad \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp\left(\alpha_j\right) & \text{if} \quad \hat{y}_j^{(i)} \neq y^{(i)} \end{cases} \qquad \text{for} \quad i = 1, 2, \cdots, m$$

سپس، تمامی وزن‌های نمونه‌ها نرمال‌سازی می‌شوند، به این صورت که هر وزن بر مجموع وزن‌های تمام نمونه‌ها $\sum_{i=1}^{m} w^{(i)}$ تقسیم می‌شود. در ادامه، یک پیش‌بینِ جدید با استفاده از وزن‌های به‌روزشده آموزش داده می‌شود و کل فرآیند تکرار می‌گردد: ابتدا وزن پیش‌بین جدید محاسبه می‌شود، سپس وزن نمونه‌ها دوباره به‌روزرسانی می‌شود و بعد پیش‌بین دیگری آموزش داده می‌شود. این روند به همین شکل ادامه می‌یابد تا زمانی که یا تعداد دلخواهی از پیش‌بین‌ها ساخته شود، یا پیش‌بینی‌کننده‌ای کامل و بدون خطا ایجاد گردد.

برای پیش‌بینی، الگوریتم AdaBoost تنها خروجی همه مدل‌ها را جمع‌آوری می‌کند و به هر یک بر اساس وزن مربوطه (α_j) وزن می‌دهد. کلاسی که بیشترین مجموع رأی‌های وزنی را دریافت کند، به‌عنوان کلاس پیش‌بینی‌شده انتخاب می‌شود (نگاه کنید به معادله ۷-۴).

© 2025 Mohsen Zare. All rights reserved.

معادله ۷-۴. پیش‌بینی‌های AdaBoost

$$\hat{y}(x) = \arg\max_k \sum_{j=1}^{N} \alpha_j \, \mathbf{1}\big(\hat{y}_j(x) = k\big)$$

که در آن:

- N ⟵ تعداد مدل‌های یادگیرنده‌ی ضعیف(پیش‌بین‌ها)
- α_j ⟵ وزن j-اُمین مدل ضعیف
- $\hat{y}_j(x)$ ⟵ پیش‌بینیِ j-اُمین مدل یادگیرنده ضعیف روی ورودی x
- $\mathbf{1}(\cdot)$ ⟵ تابع نشانگر(شاخص)، اگر شرط درست باشد برابر با ۱ و در غیر این صورت ۰ است
- k ⟵ برچسب کلاس احتمالی
- arg max ⟵ کلاس k را با ماکسیمُم رأی وزنی انتخاب می‌کند.

کتابخانه Scikit-Learn یک نسخه چندکلاسه از الگوریتم AdaBoost به نام *SAMME*، که مخفف *Stagewise Additive Modeling using a Multiclass Exponential loss function* و به معنای «مدل‌سازی افزایشی مرحله‌ای با استفاده از تابع زیان نمایی چندکلاسه» است ارائه می‌دهد. زمانی که تنها دو کلاس وجود داشته باشد، SAMME معادل AdaBoost خواهد بود. همچنین، اگر پیش‌بین‌ها قادر به تخمین احتمالات کلاس باشند—یعنی متد predict_proba() را داشته باشند—Scikit-Learn می‌تواند از نسخه دیگری به نام SAMME.R استفاده کند. حرف R مخفف "Real" به معنای «واقعی» است و در این نسخه به جای استفاده مستقیم از پیش‌بینی‌ها، الگوریتم بر پایه احتمالات کلاس عمل می‌کند که معمولاً منجر به عملکرد بهتر می‌شود.

کد زیر یک طبقه‌بند AdaBoost را با استفاده از کلاس AdaBoostClassifier از کتابخانه‌ی Scikit-Learn آموزش می‌دهد. این طبقه‌بند بر پایه‌ی ۳۰ «درخت تصمیم ساده یا درختچه‌ی تصمیم یک‌سطحی (*decision stump*)» ساخته شده است (همان‌طور که انتظار می‌رود، کلاس AdaBoostRegressor نیز برای مسائل رگرسیون موجود است). یک decision stump در

© 2025 Mohsen Zare. All rights reserved.

حقیقت درخت تصمیمی با max_depth=1 است؛ یعنی درختی شامل یک گره تصمیم و دو گره برگ. این درخت، به عنوان تخمین‌گر پایه پیش‌فرض برای کلاس AdaBoostClassifier عمل می‌کند:

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
  DecisionTreeClassifier(max_depth=1), n_estimators=30,
  learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

نکته

اگر مدل AdaBoost شما تمایل به بیش‌برازش روی مجموعه‌ی آموزشی داشته باشد، می‌توانید تعداد تخمین‌گرها را کاهش دهید یا تخمین‌گر پایه را با شدت بیشتری منظم‌سازی کنید.

© 2025 Mohsen Zare. All rights reserved.

تقویت گرادیان

یکی دیگر از الگوریتم‌های بسیار محبوب در حوزه تقویت، «تقویت گرادیان (*Gradient Boosting*)» است. مشابه AdaBoost، تقویت گرادیان با افزودن متوالی پیش‌بین‌ها به مجموعه کار می‌کند، به‌طوری که هر پیش‌بین، خطاهای پیش‌بینِ قبلی را اصلاح می‌کند. با این تفاوت که، برخلاف AdaBoost که در هر تکرار وزن نمونه‌ها را تغییر می‌دهد، در تقویت گرادیان الگوریتم تلاش می‌کند تا پیش‌بین جدید را بر اساس «خطاهای باقیمانده (*residual errors*)»ی پیش‌بین قبلی بسازد.

بیایید یک مثال ساده رگرسیون را بررسی کنیم که در آن از درخت‌های تصمیم به عنوان پیش‌بین‌های پایه استفاده می‌شود. این روش به نام «تقویت درخت گرادیان (*Gradient Tree Boosting*)» یا «درخت‌های رگرسیون تقویت‌شده با گرادیان (GBRT: *Gradient Boosted Regression Trees*)» شناخته می‌شود. ابتدا یک مجموعه داده درجه دوم نویزی ایجاد می‌کنیم و سپس یک DecisionTreeRegressor را بر روی آن برازش می‌دهیم:

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3 * X[:, 0] ** 2 + 0.05 * np.random.randn(100) # y = 3x² + Gaussian noise

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)
```

در گام بعد، یک DecisionTreeRegressor دوم روی خطاهای باقیمانده‌ی پیش‌بینِ اول آموزش داده می‌شود تا مدل بتواند خطاهای قبلی را اصلاح کند و دقت پیش‌بینی‌ها افزایش یابد:

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=43)
tree_reg2.fit(X, y2)
```

سپس یک DecisionTreeRegressor سوم روی خطاهای باقیمانده‌ی پیش‌بینِ دوم آموزش داده می‌شود تا مدل بتواند خطاهای باقی‌مانده را اصلاح کند و دقت پیش‌بینی‌ها بهبود یابد:

© 2025 Mohsen Zare. All rights reserved.

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=44)
tree_reg3.fit(X, y3)
```

اکنون ما یک مجموعه شامل سه درخت تصمیم داریم. برای پیش‌بینی روی یک نمونه جدید، کافی است پیش‌بینی‌های هر سه درخت با هم جمع شوند تا خروجی نهایی مدل به دست آید:

```
>>> X_new = np.array([[-0.4], [0.], [0.5]])
>>> sum(tree. predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
array([0.49484029, 0.04021166, 0.75026781])
```

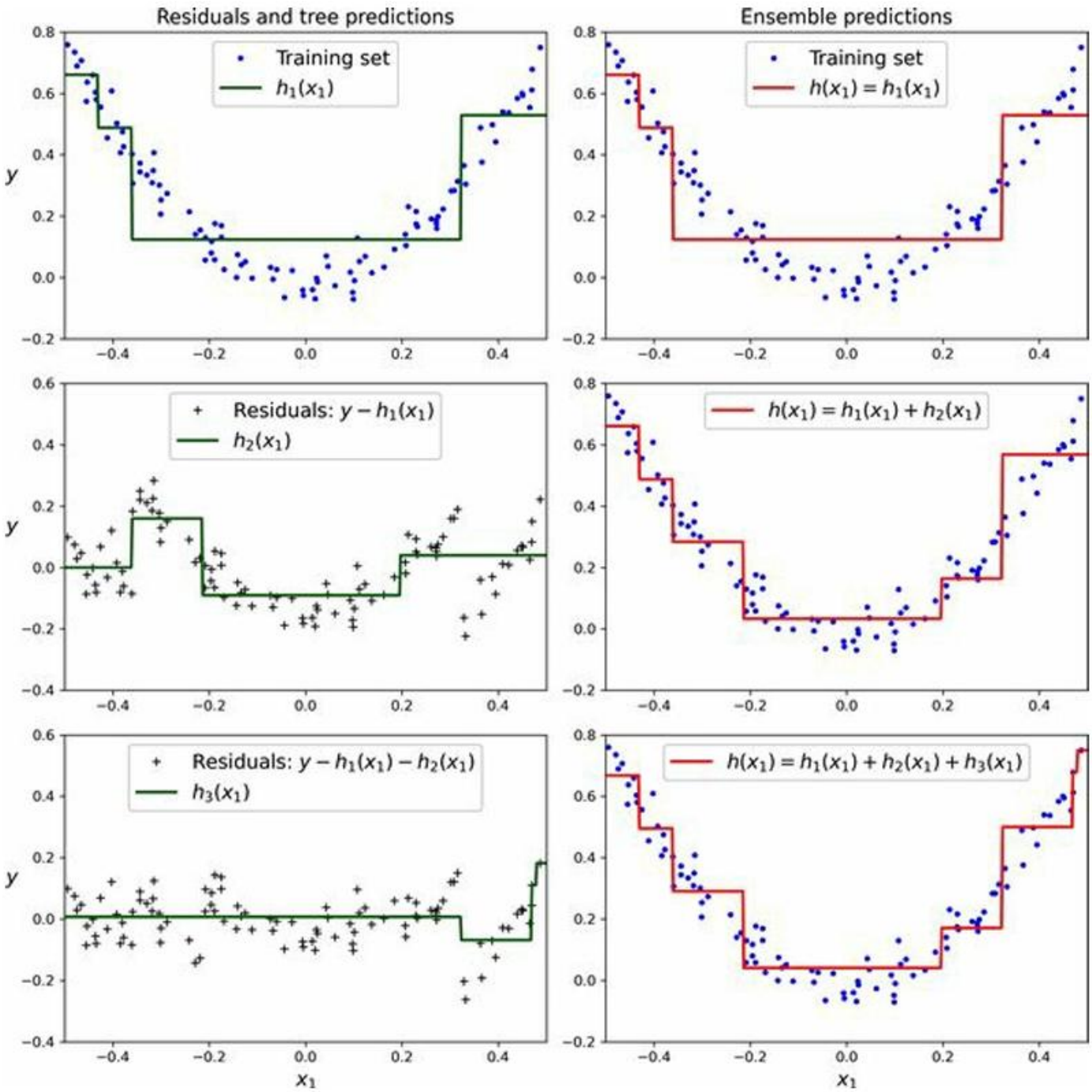
شکل ۷-۹ پیش‌بینی‌های سه درخت را نمایش می‌دهد: ستون سمت چپ مربوط به پیش‌بینی‌های تک‌تک درخت‌ها و ستون سمت راست مربوط به پیش‌بینی‌های مجموعه درخت‌ها است:

- در ردیف اول، مجموعه تنها شامل یک درخت است، بنابراین پیش‌بینی‌های گروه دقیقاً با پیش‌بینی‌های درخت اول مطابقت دارد.
- در ردیف دوم، یک درخت جدید بر اساس خطاهای باقیمانده‌ی درخت اول آموزش داده می‌شود. در سمت راست مشاهده می‌کنید که پیش‌بینی‌های گروه برابر با مجموع پیش‌بینی‌های دو درخت اول است.
- در ردیف سوم، درخت دیگری بر اساس خطاهای باقیمانده‌ی درخت دوم آموزش داده می‌شود. همان‌طور که دیده می‌شود، با افزودن هر درخت جدید به مجموعه، پیش‌بینی‌های گروه به تدریج بهبود می‌یابند.

برای سهولت در آموزش مجموعه‌های GBRT، می‌توانید از کلاس GradientBoostingRegressor در Scikit-Learn استفاده کنید. همچنین، برای مسائل طبقه‌بندی، کلاس GradientBoostingClassifier در دسترس است. این کلاس شباهت زیادی به RandomForestRegressor دارد و شامل هایپرپارامترهایی برای کنترل رشد درخت‌های تصمیم است، مانند max_depth و min_samples_leaf . همچنین، هایپرپارامترهایی برای مدیریت آموزش ترکیبی در دسترس است، از جمله تعداد درخت‌ها (n_estimators). کد زیر نمونه‌ای از ایجاد همان مجموعه قبلی را نشان می‌دهد:

© 2025 Mohsen Zare. All rights reserved.

```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3,
                learning_rate=1.0, random_state=42)
gbrt.fit(X, y)
```



شکل ۷-۹. در این تصویر از تقویت گرادیان، اولین پیش‌بین (بالا سمت چپ) به صورت معمولی آموزش داده می‌شود.

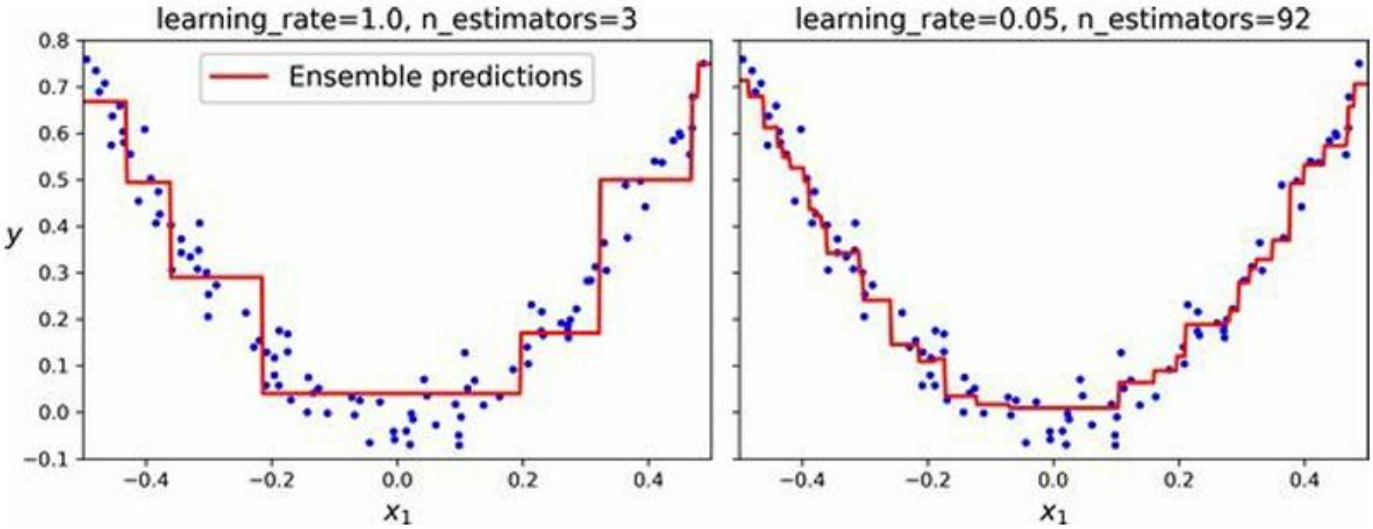
© 2025 Mohsen Zare. All rights reserved.

هر پیش‌بینِ متوالی (وسط و پایین سمت چپ) روی خطاهای پیش‌بینِ قبلی آموزش داده می‌شود. ستون سمت راست نشان می‌دهد که پیش‌بینی‌های گروه با ترکیب تمام پیش‌بین‌ها چگونه شکل می‌گیرد.

هایپرپارامتر learning_rate سهم هر درخت را مقیاس‌بندی می‌کند. اگر آن را روی مقدار کمی مانند ۰/۰۵ تنظیم کنید، برای تطبیق و برازش با مجموعه آموزشی به درخت‌های بیشتری نیاز خواهید داشت، اما معمولاً مدل پیش‌بینی‌های بهتری ارائه می‌دهد و بهتر تعمیم می‌یابد. این تکنیک نوعی منظم‌سازی به نام «انقباض (shrinkage)» است. شکل ۷-۱۰ دو مجموعه GBRT را با مقادیر مختلف هایپرپارامتر نشان می‌دهد:

- مدل سمت چپ تعداد کافی درخت ندارد و نمی‌تواند مجموعه آموزش را به‌خوبی برازش دهد.
- مدل سمت راست تقریباً تعداد مناسبی درخت دارد و پیش‌بینی‌ها بهتر تعمیم می‌یابند.

اگر تعداد درخت‌های بیشتری اضافه شود، مدل GBRT ممکن است روی داده‌های آموزش بیش‌برازش کند.



شکل ۱۰-۷. مجموعه‌های GBRT با تعداد پیش‌بین‌های ناکافی (سمت چپ) و تعداد کافی (سمت راست)

برای یافتن تعداد بهینه درخت‌ها، می‌توانید همانند گذشته از اعتبارسنجی متقاطع با استفاده از GridSearchCV یا RandomizedSearchCV بهره ببرید. با این حال، یک روش ساده‌تر نیز وجود دارد: اگر هایپرپارامتر n_iter_no_change را روی یک عدد صحیح مانند ۱۰ تنظیم کنید، کلاس GradientBoostingRegressor به‌صورت خودکار افزودن درخت‌های جدید را متوقف می‌کند، در

© 2025 Mohsen Zare. All rights reserved.

صورتی که ۱۰ درخت آخر هیچ بهبودی ایجاد نکرده باشند. این همان مفهوم توقف زودهنگام است که در فصل ۴ معرفی شد، اما با کمی انعطاف: الگوریتم چند تکرار بدون پیشرفت را تحمل می‌کند و سپس آموزش را متوقف می‌سازد. بیایید اکنون این مجموعه مدل را با استفاده از توقف زودهنگام آموزش دهیم:

```
gbrt_best = GradientBoostingRegressor(
  max_depth=2, learning_rate=0.05, n_estimators=500,
  n_iter_no_change=10, random_state=42)
gbrt_best.fit(X, y)
```

اگر مقدار n_iter_no_change خیلی کم تنظیم شود، ممکن است آموزش خیلی زود متوقف شود و مدل دچار کم‌برازش گردد. در مقابل، اگر این مقدار بیش از حد زیاد باشد، مدل ممکن است دچار بیش‌برازش شود. ما همچنین یک نرخ یادگیری نسبتاً پایین و تعداد زیادی تخمین‌گر تعیین کرده‌ایم، اما تعداد واقعی تخمین‌گرها در مجموعه آموزش‌دیده، به دلیل استفاده از توقف زودهنگام، بسیار کمتر از این مقدار اولیه است:

```
>>> gbrt_best.n_estimators_
92
```

وقتی هایپرپارامتر n_iter_no_change تنظیم شده باشد، متد fit() به‌صورت خودکار مجموعه آموزش را به دو بخش تقسیم می‌کند: یک زیرمجموعه کوچک برای آموزش و یک مجموعه اعتبارسنجی. این کار به الگوریتم اجازه می‌دهد تا پس از افزودن هر درخت جدید، عملکرد مدل را ارزیابی کند. اندازه مجموعه اعتبارسنجی توسط هایپرپارامتر validation_fraction کنترل می‌شود که مقدار پیش‌فرض آن ۱۰% است. هایپرپارامتر tol نیز بیشترین میزان بهبود عملکرد را مشخص می‌کند که هنوز بی‌اهمیت تلقی می‌شود، با مقدار پیش‌فرض ۰/۰۰۰۱.

علاوه بر این، کلاس GradientBoostingRegressor از هایپرپارامتر subsample پشتیبانی می‌کند که نسبت نمونه‌های آموزشی مورد استفاده برای آموزش هر درخت را تعیین می‌کند. برای مثال، اگر subsample=0.25 تنظیم شود، هر درخت تنها روی ۲۵% از نمونه‌های آموزشی که به صورت تصادفی انتخاب شده‌اند، آموزش داده می‌شود. همان‌طور که احتمالاً حدس زده‌اید، این تکنیک باعث افزایش بایاس و کاهش واریانس می‌شود و بنابراین نوعی موازنه ایجاد می‌کند. همچنین، فرآیند آموزش

© 2025 Mohsen Zare. All rights reserved.

را به‌طور قابل‌توجهی سریع‌تر می‌کند. این روش به نام «تقویت گرادیانی تصادفی (*Stochastic Gradient Boosting*)» شناخته می‌شود.

© 2025 Mohsen Zare. All rights reserved.

تقویت گرادیانیِ مبتنی بر هیستوگرام

کتابخانه Scikit-Learn همچنین یک پیاده‌سازی دیگر از GBRT ارائه می‌دهد که برای مجموعه‌داده‌های بزرگ بهینه‌سازی شده است: «تقویت گرادیانی مبتنی بر هیستوگرام (HGB: *Histogram-based Gradient Boosting*)». این روش بر اساس دسته‌بندیِ بازه‌ای(یا سطلی) ویژگی‌های ورودی (binning) عمل می‌کند، به این صورت که ویژگی‌ها به بازه‌های متوالی تقسیم شده و با اعداد صحیح جایگزین می‌شوند. تعداد این بخش‌ها توسط هایپرپارامتر max_bins کنترل می‌شود، که مقدار پیش‌فرض آن ۲۵۵ است و نمی‌توان آن را بالاتر از این مقدار تنظیم کرد. دسته‌بندیِ بازه‌ای مزایای زیادی دارد:

- تعداد آستانه‌های ممکن که الگوریتم باید در طول آموزش ارزیابی کند، به‌طور قابل‌توجهی کاهش می‌یابد.
- استفاده از اعداد صحیح امکان بهره‌گیری از ساختارهای داده‌ای سریع‌تر و حافظه‌بهره‌وری بالاتر را فراهم می‌کند.
- روش ساخت دسته‌بندی‌های سطلی باعث می‌شود نیاز به مرتب‌سازی ویژگی‌ها هنگام آموزش هر درخت برطرف شود، که سرعت آموزش را افزایش می‌دهد.

در نتیجه، این پیاده‌سازی دارای پیچیدگی محاسباتی $O(b \times m)$ است، در حالی که نسخه معمولی GBRT دارای پیچیدگی$O(n \times m \times \log(m))$ است، که در اینجا: b تعداد بازه‌ها یا سطل‌ها (bins)، m تعداد نمونه‌های آموزشی ، و n تعداد ویژگی‌ها است. این بدان معناست که الگوریتم HGB می‌تواند روی مجموعه داده‌های بزرگ صدها برابر سریع‌تر از GBRT معمولی آموزش یابد. با این حال، بازه‌بندی سطلی ممکن است موجب کاهش دقت (precision loss) شود و به نوعی همانند یک تکنیک منظم‌سازی عمل کند. بسته به ویژگی‌های داده‌ها، این موضوع می‌تواند:

- بیش‌برازش را کاهش دهد و عملکرد مدل روی داده‌های جدید را بهبود بخشد، یا
- باعث کم‌برازش شود و مدل نتواند الگوهای موجود در داده‌ها را به‌خوبی بیاموزد.

© 2025 Mohsen Zare. All rights reserved.

کتابخانه Scikit-Learn دو کلاس برای گرادیان بوستینگ مبتنی بر هیستوگرام (HGB) ارائه می‌دهد:

- HistGradientBoostingRegressor برای مسائل رگرسیون (پیش‌بینی مقادیر عددی)
- HistGradientBoostingClassifier برای مسائل طبقه‌بندی (پیش‌بینی برچسب‌های دسته‌ای)

این کلاس‌ها در ظاهر شباهت‌هایی با کلاس‌های GradientBoostingRegressor و GradientBoostingClassifier دارند، اما با چند تفاوت قابل توجه:

- اگر تعداد نمونه‌ها بیش از ۱۰,۰۰۰ باشد، توقف زودهنگام (early stopping) به‌صورت خودکار فعال می‌شود. همچنین می‌توانید با تنظیم هایپرپارامتر early_stopping روی True یا False، توقف زودهنگام را همیشه فعال یا همیشه غیرفعال کنید.
- زیرنمونه‌گیری (subsampling) یا نمونه‌گیری زیرمجموعه‌ای پشتیبانی نمی‌شود.
- هایپرپارامتر n_estimators به max_iter تغییر نام داده است.
- تنها هایپرپارامترهای درخت تصمیم‌گیری که قابل تنظیم هستند عبارت‌اند از:
  - max_leaf_nodes: حداکثر تعداد گره‌های برگ
  - min_samples_leaf: حداقل تعداد نمونه در هر برگ
  - max_depth: حداکثر عمق درخت

کلاس‌های HGB دو ویژگی مهم دارند: آن‌ها هم از ویژگی‌های دسته‌ای (categorical features) پشتیبانی می‌کنند و هم از مقادیر مفقود (missing values). این قابلیت‌ها فرآیند پیش‌پردازش داده‌ها را تا حد زیادی ساده می‌کنند. با این حال، ویژگی‌های دسته‌ای باید به صورت اعداد صحیح از ۰ تا مقداری کمتر از max_bins نمایش داده شوند. برای این کار می‌توانید از OrdinalEncoder استفاده کنید. برای مثال، در ادامه نحوه‌ی ساخت و آموزش یک پایپلاین کامل برای مجموعه داده مسکن کالیفرنیا که در فصل ۲ معرفی شد، نشان داده شده است:

```
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
```

© 2025 Mohsen Zare. All rights reserved.

```
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.preprocessing import OrdinalEncoder

hgb_reg = make_pipeline(
  make_column_transformer((OrdinalEncoder(), ["ocean_proximity"]),
          remainder="passthrough"),
  HistGradientBoostingRegressor(categorical_features=[0], random_state=42)
)
hgb_reg.fit(housing, housing_labels)
```

کل پایپلاین به‌قدری کوتاه است که تقریباً همه چیز را می‌توان بدون نیاز به وارد کردن کتابخانه‌های اضافی انجام داد! نیازی به برآوردگر و جایگذار داده های مفقود (imputer)، مقیاس‌گذار (scaler) یا کدگذار وان-هات (one-hot encoder) نیست، بنابراین استفاده از آن بسیار ساده و کاربردی است. توجه داشته باشید که categorical_features باید به صورت اندیس ستون‌های دسته‌ای یا یک آرایه بولی مشخص شود. بدون هیچ گونه تنظیم خاصی برای هایپرپارامترها، این مدل به مقدار RMSE حدود ۴۷,۶۰۰ دست می‌یابد، که نتیجه قابل قبولی محسوب می‌شود.

نکته

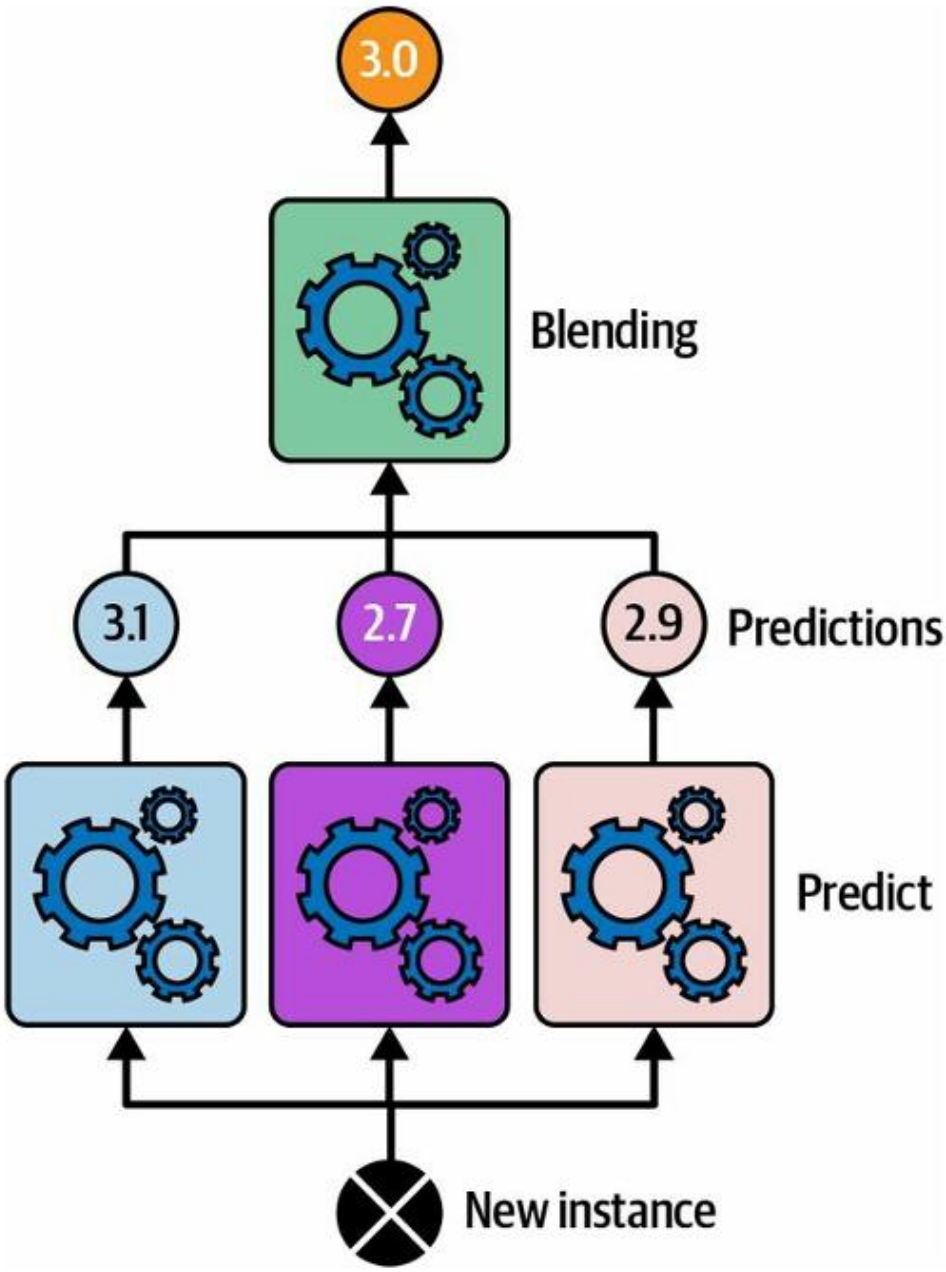
چندین پیاده‌سازی بهینه‌شده دیگر از الگوریتم تقویت گرادیان در اکوسیستم یادگیری ماشین پایتون وجود دارد؛ به‌ویژه: XGBoost، CatBoost، LightGBM. این کتابخانه‌ها چند سالی است که توسعه یافته‌اند و به طور خاص برای تقویت گرادیان طراحی شده‌اند. API این کتابخانه‌ها بسیار شبیه به Scikit-Learn است و ویژگی‌های اضافی فراوانی دارند، از جمله پشتیبانی از شتاب‌دهی با GPU؛ بنابراین، بررسی آن‌ها قطعاً ارزشمند است. علاوه بر این، کتابخانه TensorFlow Random Forests نیز پیاده‌سازی‌های بهینه‌شده‌ای از انواع الگوریتم‌های جنگل تصادفی ارائه می‌دهد، از جمله: plain random forests ، Extra-Trees، GBRT و چند الگوریتم دیگر.

© 2025 Mohsen Zare. All rights reserved.

یادگیری پُشته‌ای(انباشتی)

«انباشت(روی‌هم‌چینی)مدل‌ها یا اِستَکینگ (Stacking)»، که کوتاه‌شده عبارت « stacked generalization به معنی تعمیم پشته‌ای(روی‌هم‌چینی تعمیم‌یافته)» است، آخرین روش تجمیعی است که در این فصل بررسی می‌کنیم. این روش بر اساس یک ایده ساده شکل گرفته است: به جای استفاده از توابع ساده مانند رأی‌گیری اکثریت(hard voting) برای تجمیع پیش‌بینی‌های تمام مدل‌های یک گروه، چرا مدلی مستقل برای انجام این تجمیع آموزش ندهیم؟ شکل ۱۱-۷ این نوع از روش تجمیعی را نشان می‌دهد که یک کار رگرسیون را روی یک نمونه جدید انجام می‌دهد. هر یک از سه پیش‌بین در پایین شکل، مقدار متفاوتی پیش‌بینی می‌کنند (۳٫۱، ۲٫۷ و ۲٫۹). سپس پیش‌بین نهایی، که به آن «مدل ترکیب‌گر (blender)» یا یک «مدل فرایادگیرنده یا مدل سطح بالاتر (meta learner)»–مدلی که خروجی‌های Base Learners یا مدل پایه را می‌گیرد و تصمیم نهایی را می‌سازد–گفته می‌شود، این پیش‌بینی‌ها را به عنوان ورودی دریافت کرده و پیش‌بینی نهایی را محاسبه می‌کند (۳٫۰).

© 2025 Mohsen Zare. All rights reserved.

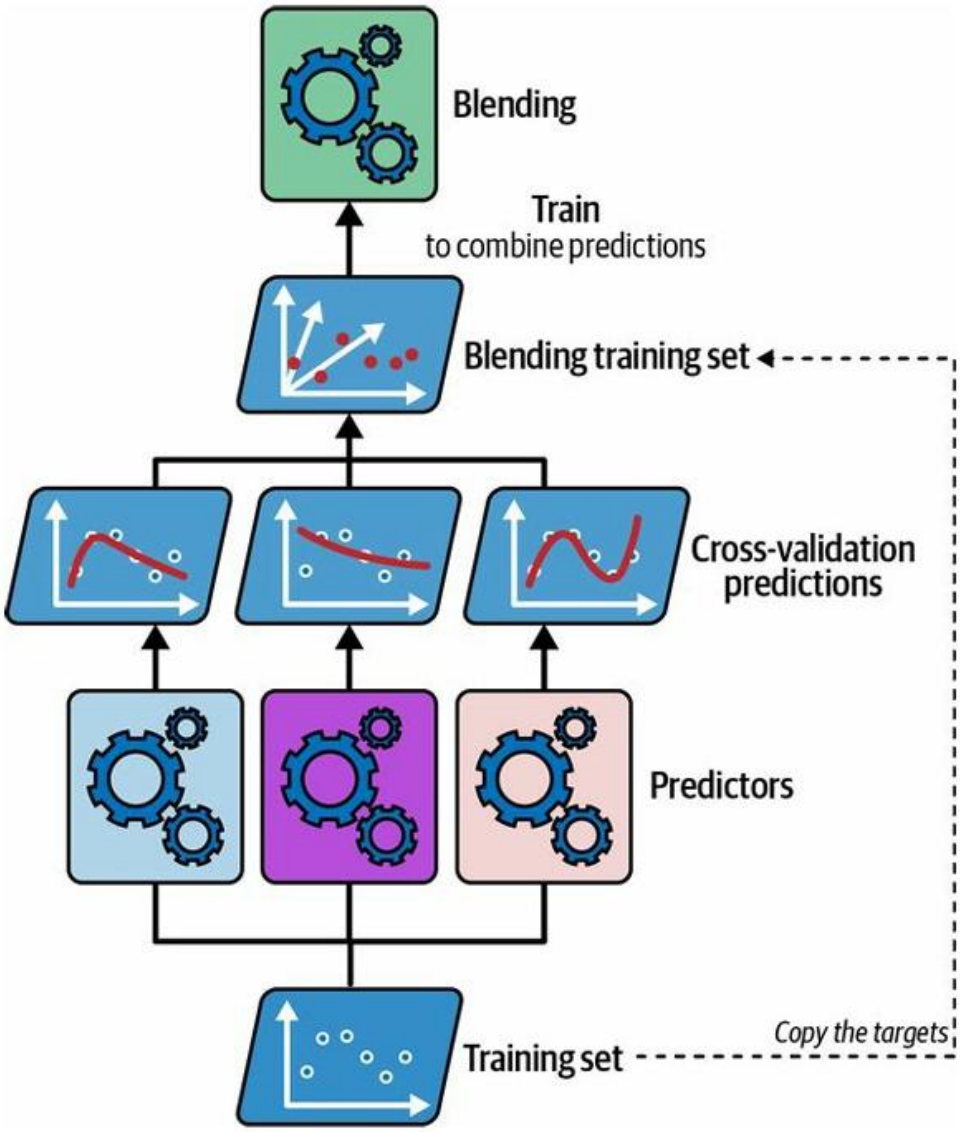


شکل ۷-۱۱. تجمیع پیش‌بینی‌ها با استفاده از یک پیش‌بینِ ترکیب‌گر (blending predictor)

برای آموزش مدل ترکیب‌گر، ابتدا باید مجموعه داده‌ی آموزشی برای مدل ترکیب‌گر را بسازید. می‌توانید از تابع ()cross_val_predict برای هر پیش‌بینی‌گر در مجموعه استفاده کنید تا پیش‌بینی‌های خارج از نمونه (out-of-sample) برای هر نمونه در مجموعه داده‌ی آموزشی اصلی به‌دست آید، همان‌طور که در شکل ۷-۱۲ نشان داده شده است. این پیش‌بینی‌ها سپس به‌عنوان ویژگی‌های ورودی برای آموزش ترکیب‌گر استفاده می‌شوند و هدف‌ها (targets) را می‌توان مستقیماً از مجموعه آموزشی اصلی کپی کرد. توجه داشته باشید که صرف‌نظر از تعداد ویژگی‌ها در مجموعه داده‌ی اصلی (که در این مثال فقط

© 2025 Mohsen Zare. All rights reserved.

یکی است)، مجموعه داده‌ی آموزشی ترکیب‌گر برای هر پیش‌بین، تنها یک ویژگی ورودی خواهد داشت؛ بنابراین در این مثال با سه پیش‌بینی‌گر، مجموعه داده ترکیب‌گر دارای سه ویژگی خواهد بود. پس از آموزش ترکیب‌گر، پیش‌بینی‌گرهای پایه یک‌بار دیگر، این بار روی کل مجموعه داده‌ی آموزشی اصلی آموزش داده می‌شوند.

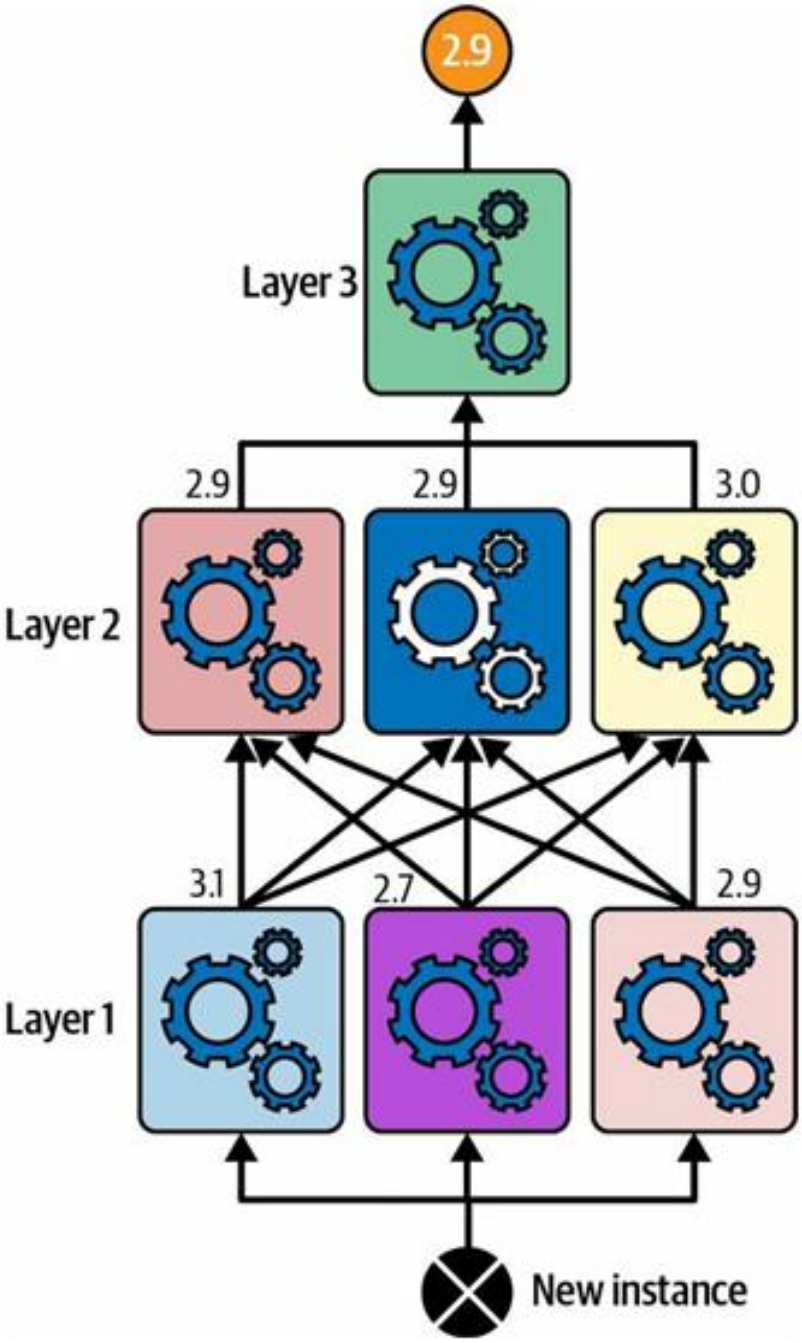


شکل ۱۲-۷. آموزش ترکیب‌گر در یک مجموعه‌ی پُشته‌ای (stacking ensemble)

در واقع، می‌توان چندین ترکیب‌گر مختلف را به این روش آموزش داد؛ برای مثال، یکی با استفاده از رگرسیون خطی و دیگری با استفاده از رگرسیون جنگل تصادفی. این کار یک لایه کامل از ترکیب‌گرها ایجاد می‌کند و سپس می‌توان یک ترکیب‌گر نهایی در بالای آن‌ها اضافه کرد تا پیش‌بینی نهایی تولید شود، همان‌طور که در شکل ۷-۱۳ نشان داده شده است.

© 2025 Mohsen Zare. All rights reserved.

انجام این کار ممکن است عملکرد مدل را کمی بهبود دهد، اما هزینه آن افزایش زمان آموزش و پیچیدگی سیستم خواهد بود.



شکل ۷-۱۳. پیش‌بینی‌ها در یک مجموعه پشته‌ای چندلایه (multilayer stacking ensemble)

کتابخانه Scikit-Learn دو کلاس برای مجموعه‌های پشته‌ای (stacking ensembles) ارائه می‌دهد:

© 2025 Mohsen Zare. All rights reserved.

- StackingClassifier
- StackingRegressor

برای مثال، می‌توانیم به جای VotingClassifier که در ابتدای این فصل روی داده‌های moons استفاده شد، از StackingClassifier بهره ببریم:

```
from sklearn.ensemble import StackingClassifier

stacking_clf = StackingClassifier(
  estimators=[
    ('lr', LogisticRegression(random_state=42)),
    ('rf', RandomForestClassifier(random_state=42)),
    ('svc', SVC(probability=True, random_state=42))
  ],
  final_estimator=RandomForestClassifier(random_state=43),
  cv=5 # number of cross-validation folds
)
stacking_clf.fit(X_train, y_train)
```

برای هر پیش‌بینی‌گر، کلاس StackingClassifier در صورتی که موجود باشد، متد predict_proba() را فراخوانی می‌کند. در غیر این صورت، به ترتیب ابتدا decision_function() و اگر آن هم موجود نباشد، predict() را صدا می‌زند. اگر یک تخمین‌گر نهایی (final estimator) ارائه ندهید، StackingClassifier به‌طور پیش‌فرض از رگرسیون لجستیک و StackingRegressor از RidgeCV استفاده خواهد کرد.

اگر این مدل استکینگ را روی مجموعه آزمون ارزیابی کنید، به دقت ۹۲٫۸% خواهید رسید، که کمی بهتر از طبقه‌بند رأی‌گیری با رأی‌دهی نرم است، که پیش‌تر دقت ۹۲% به دست آورده بود.

در جمع‌بندی، روش‌های ensemble (تجمیعی) ابزارهایی قدرتمند، چندمنظوره و نسبتاً ساده برای استفاده در یادگیری ماشین هستند. مدل‌هایی مانند جنگل تصادفی، AdaBoost و گرادیان بوستینگ (GBRT) از جمله نخستین مدل‌هایی هستند که در اکثر مسائل یادگیری ماشین باید امتحان شوند، به‌ویژه زمانی که با داده‌های جدولی و ناهمگن (heterogeneous tabular data) سر و کار دارید. این مدل‌ها به پیش‌پردازش کمی نیاز دارند و بنابراین برای ساخت نمونه اولیه سریع (prototype)

© 2025 Mohsen Zare. All rights reserved.

بسیار مناسب‌اند. در نهایت، روش‌های گروهی مانند طبقه‌بندهای رأی‌گیری و طبقه‌بندهای انباشتی (stacking) می‌توانند عملکرد مدل شما را تا حد امکان بهبود دهند.

© 2025 Mohsen Zare. All rights reserved.

تمرین‌ها

۱. اگر پنج مدل مختلف را روی دقیقاً یک داده‌ی آموزشی آموزش داده باشید و همگی به دقت ۹۵% دست یافته باشند، آیا امکان دارد با ترکیب این مدل‌ها نتایج بهتری بگیرید؟ اگر بله، چگونه؟ اگر نه، چرا؟

۲. تفاوت بین دسته‌بندهای رأی‌گیری سخت و رأی‌گیری نرم چیست؟

۳. آیا امکان دارد با توزیع آموزش یک مجموعه Bagging روی چندین سرور، سرعت آموزش را افزایش داد؟ در مورد مجموعه‌های Pasting، Boosting، جنگل‌های تصادفی یا مجموعه‌های Stacking چطور؟

۴. مزیت ارزیابی خارج از بسته (Out-of-Bag Evaluation) چیست؟

۵. چه چیزی باعث می‌شود گروه‌های Extra-Trees نسبت به جنگل‌های تصادفی معمولی تصادفی‌تر باشند؟ این تصادفی بودن بیشتر چگونه می‌تواند مفید باشد؟ آیا دسته‌بندهای Extra-Trees نسبت به جنگل‌های تصادفی معمولی کندتر هستند یا سریع‌تر؟

۶. اگر مدل گروهی AdaBoost شما داده‌های آموزشی را کم‌برازش می‌کند، باید کدام هایپرپارامترها را تنظیم کنید و چگونه؟

۷. اگر گروهِ گرادیان بوستینگ شما روی داده‌های آموزشی بیش‌برازش می‌کند، آیا باید نرخ یادگیری (learning rate) را افزایش دهید یا کاهش؟

۸. مجموعه داده MNIST (معرفی شده در فصل ۳) را بارگذاری کنید و آن را به سه مجموعه تقسیم کنید: مجموعه آموزشی، مجموعه اعتبارسنجی و مجموعه آزمون (مثلاً ۵۰,۰۰۰ نمونه برای آموزش، ۱۰,۰۰۰ برای اعتبارسنجی و ۱۰,۰۰۰ برای آزمون). سپس چند دسته‌بند مختلف مانند Random Forest، Extra-Trees و SVM را آموزش دهید. بعد تلاش کنید آن‌ها را با استفاده از رأی‌گیری نرم یا سخت ترکیب کنید تا گروهی بسازید که روی مجموعه اعتبارسنجی

© 2025 Mohsen Zare. All rights reserved.

بهتر از هر دسته‌بند جداگانه عمل کند. پس از پیدا کردن چنین گروهی، آن را روی مجموعه آزمون امتحان کنید .عملکرد آن نسبت به دسته‌بندهای جداگانه چقدر بهتر است؟

۹. دسته‌بندهای جداگانه‌ای که در تمرین قبلی آموزش داده‌اید را اجرا کنید تا روی مجموعه اعتبارسنجی پیش‌بینی انجام دهند. سپس یک مجموعه آموزش جدید بسازید که از این پیش‌بینی‌ها تشکیل شده باشد: هر نمونه آموزشی یک بردار است که شامل مجموعه‌ای از پیش‌بینی‌های همهٔ دسته‌بندهای شما برای یک تصویر است، و برچسب هدف آن، کلاس واقعی تصویر خواهد بود. یک دسته‌بند را روی این مجموعه‌دادهٔ آموزشی جدید آموزش دهید. تبریک می‌گویم – شما اکنون یک «ترکیب‌گر (blender)» را آموزش داده‌اید، و همراه با دسته‌بندهای قبلی، یک «گروه پشته‌ای» پساخته‌اید! اکنون این مجموعهٔ پشته‌ای را روی مجموعهٔ آزمون ارزیابی کنید. برای هر تصویر در مجموعهٔ آزمون، با تمام دسته‌بندهای خود پیش‌بینی انجام دهید، سپس این پیش‌بینی‌ها را به ترکیب‌گر بدهید تا پیش‌بینی نهایی گروه پشته‌ای را به دست آورید. مقایسه کنید که این روش چه تفاوتی با دسته‌بند رأی‌گیری (voting classifier) دارد که قبلاً آموزش داده بودید؟ اکنون همین فرآیند را با استفاده از کلاس StackingClassifier در Scikit-Learn تکرار کنید. آیا عملکرد بهتری به دست می‌آورید؟ اگر بله، چرا؟

پاسخ‌های مربوط به این تمرین‌ها را می‌توانید در انتهای نوت‌بوک همین فصل، از طریق نشانی *https://homl.info/colab3* مشاهده کنید.

© 2025 Mohsen Zare. All rights reserved.

فصل ۸. کاهش ابعاد

بسیاری از مسائل یادگیری ماشین شامل هزاران یا حتی میلیون‌ها ویژگی برای هر نمونه‌ی آموزشی هستند. این حجم زیاد ویژگی‌ها نه‌تنها فرآیند آموزش را بسیار کند می‌کند، بلکه یافتن یک راه‌حل مناسب را نیز به‌طور قابل‌توجهی دشوار می‌سازد، همان‌طور که در ادامه خواهید دید. این مشکل اغلب با عنوان «نفرین بُعد بالا (*curse of dimensionality*)»–به مفهوم مُعضل ابعادی؛ که توضیح می‌دهد چرا در داده‌های با ویژگی‌های زیاد (high-dimensional data) یادگیری ماشین با مشکل مواجه می‌شود– شناخته می‌شود.

خوشبختانه، در مسائل دنیای واقعی، اغلب می‌توان تعداد ویژگی‌ها را به‌طور قابل‌توجهی کاهش داد و یک مسئله‌ی غیرقابل‌حل را به مسئله‌ای قابل‌حل تبدیل کرد. برای مثال، به تصاویر MNIST(که در فصل ۳ معرفی شدند) توجه کنید: پیکسل‌های لبه‌ی تصاویر تقریباً همیشه سفید هستند، بنابراین می‌توانید این پیکسل‌ها را به‌طور کامل از مجموعه‌ی آموزشی حذف کنید بدون اینکه اطلاعات زیادی را از دست بدهید. همان‌طور که در فصل قبل دیدیم، شکل ۶-۷ تأیید می‌کند که این پیکسل‌ها کاملاً بی‌اهمیت برای کار طبقه‌بندی هستند. علاوه بر این، دو پیکسل مجاور اغلب به‌شدت با یکدیگر هم‌بسته‌اند: اگر آن‌ها را در یک پیکسل واحد ادغام کنید (مثلاً با گرفتن میانگین شدت روشنایی دو پیکسل)، اطلاعات چندانی از دست نخواهید داد.

© 2025 Mohsen Zare. All rights reserved.

هشدار

کاهش ابعاد، مقداری از اطلاعات را از بین می‌برد؛ درست مانند فشرده‌سازی یک تصویر به فرمت JPEG که می‌تواند کیفیت آن را کاهش دهد. بنابراین، با اینکه این کار آموزش مدل را سریع‌تر می‌کند، اما ممکن است عملکرد سیستم شما را اندکی بدتر کند. همچنین باعث پیچیده‌تر شدن جریان پردازش داده‌ها (pipeline) می‌شود و در نتیجه نگهداری آن را دشوارتر می‌سازد. از این رو، توصیه می‌کنم ابتدا سعی کنید سیستم خود را با داده‌های اصلی آموزش دهید، پیش از آن‌که به استفاده از کاهش ابعاد فکر کنید. در برخی موارد، کاهش ابعاد داده‌های آموزشی ممکن است مقداری از نویزها و جزئیات غیرضروری را فیلتر کند و در نتیجه عملکرد را افزایش دهد، اما به‌طور کلی این اتفاق نمی‌افتد؛ فقط باعث افزایش سرعت آموزش می‌شود.

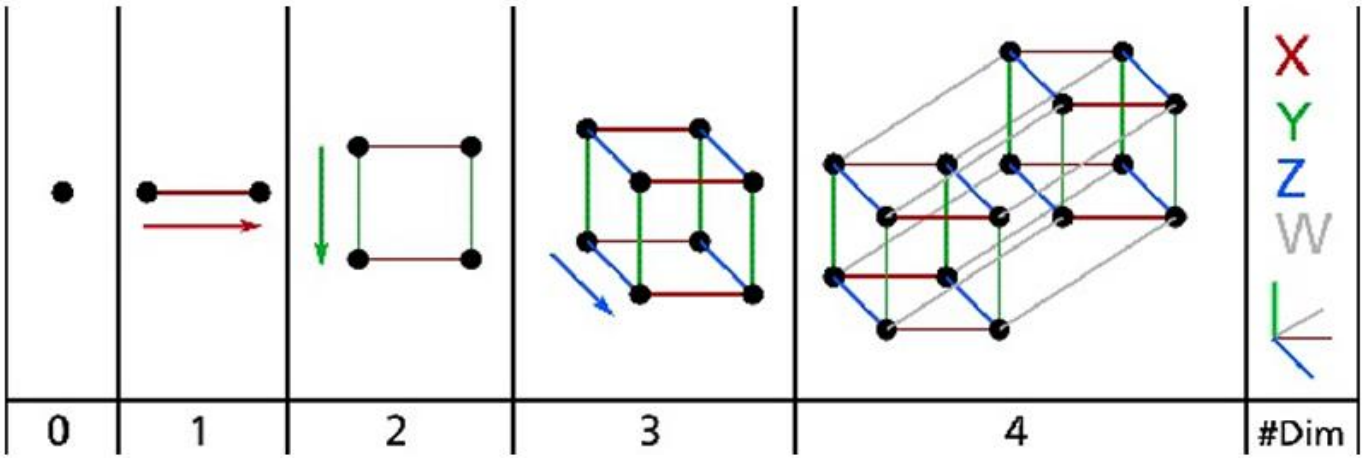
علاوه بر افزایش سرعت آموزش، کاهش ابعاد برای مصورسازی داده‌ها نیز بسیار مفید است .با کاهش تعداد ابعاد به دو (یا سه) بُعد، می‌توان نمایی فشرده از یک مجموعه داده با ابعاد بالا را روی یک نمودار ترسیم کرد و اغلب با مشاهده الگوها مانند خوشه‌ها، می‌توان به اطلاعات و بینش‌های مهمی دست یافت. علاوه بر این، مصورسازی داده‌ها برای انتقال نتایج به افرادی که دانشمند داده نیستند نیز بسیار حیاتی است—به‌ویژه تصمیم‌گیرندگانی که قرار است از نتایج شما استفاده کنند.

در این فصل ابتدا درباره‌ی نفرین بُعد بالا صحبت خواهیم کرد و درک بهتری از آنچه در فضای با ابعاد بالا رخ می‌دهد به‌دست می‌آوریم. سپس دو روش اصلی برای کاهش ابعاد را بررسی خواهیم کرد: «پروجکشن یا تصویرنگاری(projection)»–انتقال داده‌ها به زیرفضای کم‌بعدی–و «یادگیری مَنیفُلد یا زیرسطح/زیرفضا (manifold learning)». در ادامه، سه مورد از محبوب‌ترین تکنیک‌های کاهش ابعاد را مرور خواهیم کرد: تحلیل مؤلفه‌های اصلی (PCA)، تصویرنگاری تصادفی (random projection) و جاسازی خطی محلی (Locally Linear Embedding یا LLE).

© 2025 Mohsen Zare. All rights reserved.

نفرین بُعد بالا

ما آن‌قدر به زندگی در فضای ۳بُعدی عادت داریم که شهودمان هنگام تصور فضاهای با ابعاد بالا از کار می‌افتد. حتی تصور یک ابرمکعب ساده‌ی چهاربعدی نیز در ذهن بسیار دشوار است (نگاه کنید به شکل ۸-۱)، چه برسد به یک بیضی‌شکل ۲۰۰بعدی که در فضایی با ۱,۰۰۰ بُعد پیچ‌خورده باشد.



شکل ۱-۸. نقطه، پاره‌خط، مربع، مکعب، و مکعب چهاربُعدی(tesseract) (ابرمکعب‌ها از صفر بُعد تا چهار بُعد)

مشخص شده است که بسیاری از چیزها در فضای با ابعاد بالا رفتار بسیار متفاوتی دارند. برای مثال، اگر یک نقطه‌ی تصادفی در یک مربع واحد (مربع ۱ × ۱) انتخاب کنید، تنها حدود ۰/۴ % احتمال دارد که این نقطه در فاصله‌ی کمتر از ۰/۰۰۱ از یک مرز قرار داشته باشد (به عبارت دیگر، بسیار بعید است که یک نقطه تصادفی در هر بعدی در موقعیتی «نهایتی» (extreme) قرار گیرد). اما در یک ابرمکعب واحد با ۱۰,۰۰۰ بعد، این احتمال بیش از ۹۹/۹۹۹۹۹۹٪ است. یعنی اکثر نقاط در یک ابرمکعب با ابعاد بالا بسیار نزدیک به لبه‌ها قرار دارند.

در اینجا یک تفاوت دردسرسازتر وجود دارد: اگر دو نقطه را به‌صورت تصادفی در یک مربع واحد انتخاب کنید، فاصله‌ی میان این دو نقطه به طور متوسط تقریباً برابر با ۰/۵۲ خواهد بود. اگر دو نقطه تصادفی در یک مکعب واحد سه‌بعدی انتخاب کنید، میانگین فاصله بین آن‌ها تقریباً ۰/۶۶ خواهد بود. اما در موردِ دو نقطه تصادفی در یک ابرمکعب واحد با ۱,۰۰۰,۰۰۰ بعد چه؟ باور کنید یا نه، میانگین فاصله تقریباً برابر با ۴۰۸/۲۵ است (حدود ریشه ششم یک میلیون)! این موضوع خلاف شهود است: چگونه ممکن

© 2025 Mohsen Zare. All rights reserved.

است دو نقطه این‌قدر از هم دور باشند در حالی که هر دو داخل یک ابرمکعب واحد قرار دارند؟ پاسخ این است که در ابعاد بالا فضای بسیار زیادی وجود دارد. در نتیجه، داده‌های با ابعاد بالا در معرض خطر پراکندگی بسیار زیاد هستند؛ به طوری که بیشتر نمونه‌های آموزشی احتمالاً از یکدیگر فاصله زیادی دارند. این موضوع همچنین به این معناست که یک نمونه جدید احتمالاً فاصله زیادی از هر نمونه‌ی آموزشی خواهد داشت، و این باعث می‌شود پیش‌بینی‌ها نسبت به حالت‌های با ابعاد پایین‌تر، کمتر قابل‌اعتماد باشند، زیرا پیش‌بینی‌ها بر پایه تخمین‌های بسیار بزرگ‌تری (extrapolation) انجام می‌شوند. به طور خلاصه، هرچه مجموعه‌ی آموزشی ابعاد بیشتری داشته باشد، ریسک بیش‌برازش نیز بیشتر خواهد بود.

از نظر تئوری، یک راه‌حل برای مقابله با «مُعضل ابعاد بالا» می‌تواند افزایش اندازه مجموعه‌ی آموزشی باشد تا به چگالی کافی از نمونه‌های آموزشی دست یابیم. متأسفانه در عمل، تعداد نمونه‌های آموزشی موردنیاز برای رسیدن به یک چگالی مشخص، به‌صورت نمایی با افزایش تعداد ابعاد رشد می‌کند. فقط با ۱۰۰ ویژگی—که بسیار کمتر از تعداد ویژگی‌ها در مسئله‌ی MNIST است—و فرض اینکه همه‌ی این ویژگی‌ها در بازه‌ی ۰ تا ۱ قرار دارند، برای اینکه نمونه‌های آموزشی به‌طور متوسط در فاصله‌ی کمتر از ۰/۱ از یکدیگر باشند (با فرض پراکندگی یکنواخت در تمام ابعاد)، به تعداد نمونه‌هایی بیش از تعداد اتم‌های موجود در جهان قابل مشاهده نیاز خواهید داشت.

© 2025 Mohsen Zare. All rights reserved.

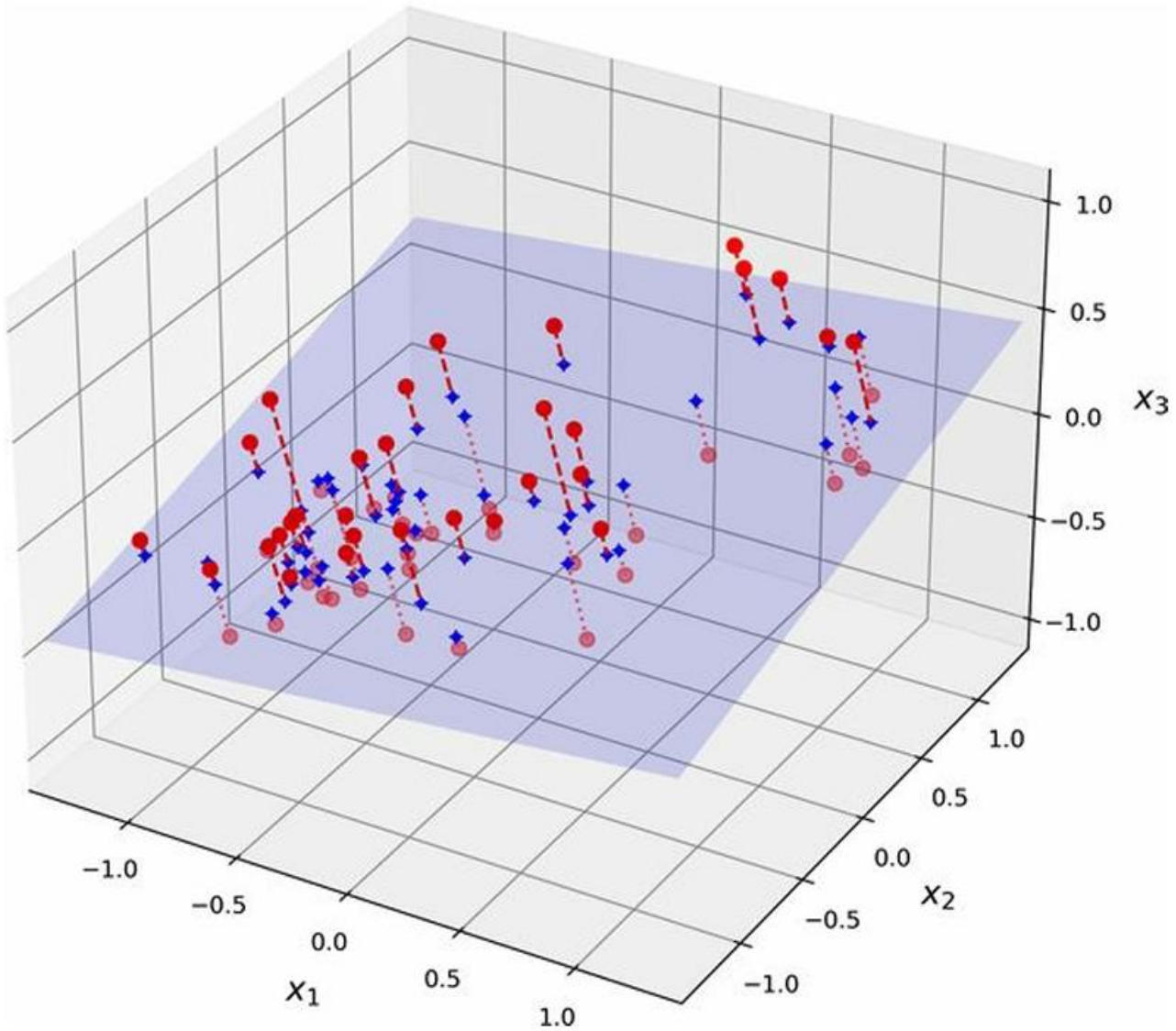
رویکردهای اصلی برای کاهش ابعاد

قبل از آن‌که وارد الگوریتم‌های خاص کاهش ابعاد شویم، بیایید نگاهی به دو رویکرد اصلی در کاهش ابعاد بیندازیم: تصویرنگاری (Projection) و یادگیری مَنیفُلد (Manifold Learning).

© 2025 Mohsen Zare. All rights reserved.

تصویرنگاری(پروژکشن)

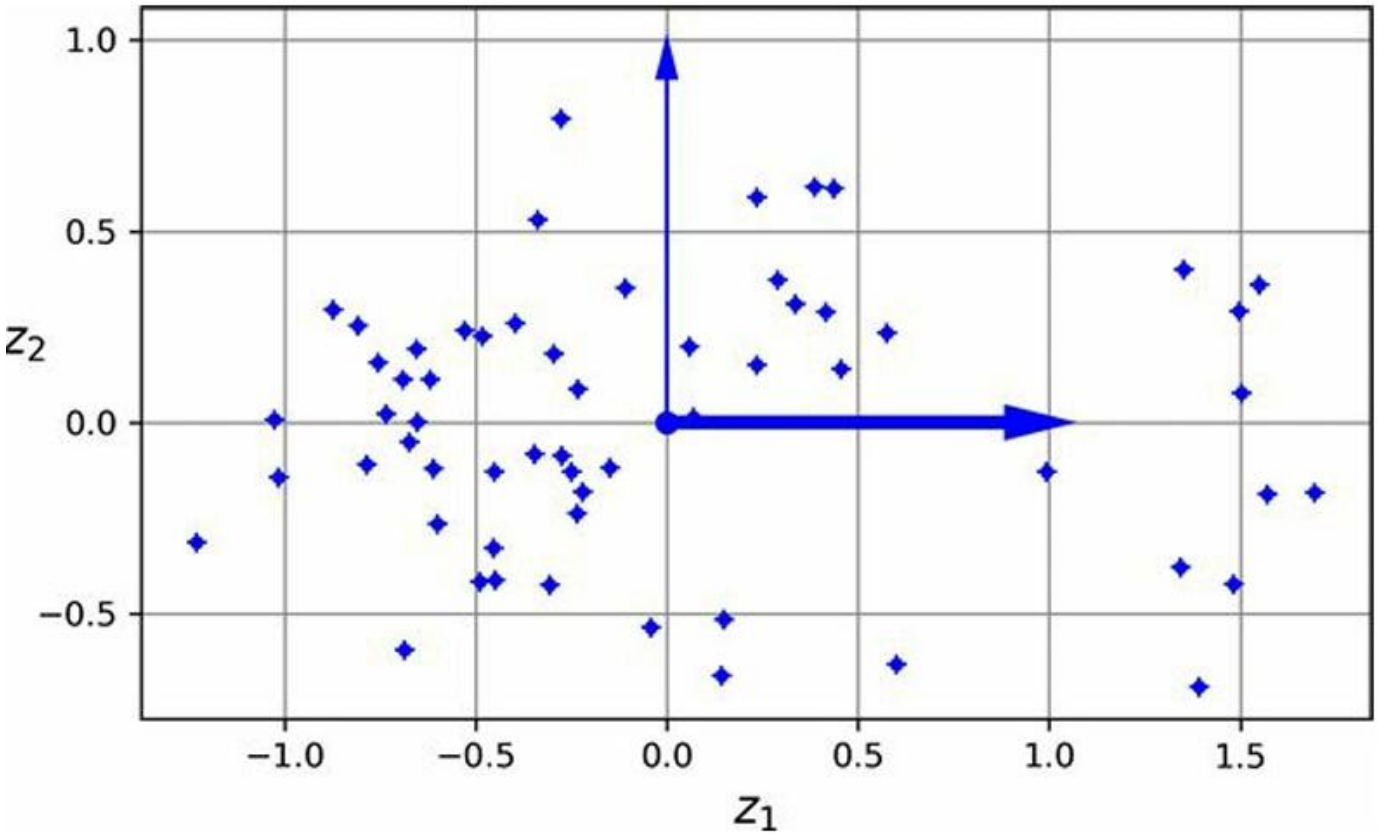
در بیشتر مسائل دنیای واقعی، نمونه‌های آموزشی به‌صورت یکنواخت در تمامی ابعاد پخش نشده‌اند. بسیاری از ویژگی‌ها تقریباً ثابت هستند، در حالی که برخی دیگر به‌شدت با یکدیگر هم‌بسته‌اند (همان‌طور که قبلاً در مورد مجموعه داده MNIST بحث شد). در نتیجه، تمام نمونه‌های آموزشی درون (یا نزدیک به) یک «زیرفضا($subspace$)»ی با بُعد بسیار کمتر از فضای بُعدبالا قرار دارند. این موضوع ممکن است بسیار انتزاعی به نظر برسد، بنابراین بیایید به یک مثال نگاه کنیم. در شکل ۸-۲، یک مجموعه‌داده‌ی سه‌بعدی را مشاهده می‌کنید که با کره‌های کوچک نمایش داده شده‌اند.



شکل ۸-۲. یک مجموعه‌داده سه‌بعدی که در نزدیکی یک زیرفضای دوبعدی قرار دارد

© 2025 Mohsen Zare. All rights reserved.

توجه کنید که تمام نمونه‌های آموزش در نزدیکی یک صفحه قرار دارند: این صفحه یک زیرفضای با بُعد کمتر (۲بعدی) از فضای با بُعد بالاتر (۳بعدی) است. اگر هر نمونه آموزشی را به‌صورت عمودی روی این زیرفضای کم‌بعد بازتاب یا پروجکت کنیم (همان‌طور که با خطوط کوتاه خط‌چین که نمونه‌ها را به صفحه متصل می‌کنند نشان داده شده است)، مجموعه‌داده جدید دوبعدی شکل ۸-۳ به‌دست می‌آید. خب! ما به‌تازگی ابعاد مجموعه‌داده را از ۳ بعد به ۲ بعد کاهش دادیم. توجه داشته باشید که محورهای نمودار به ویژگی‌های جدید z_1 و z_2 مربوط می‌شوند؛ این‌ها مختصات تصویرنگاری‌ها(پروجکشن‌ها) روی صفحه هستند.

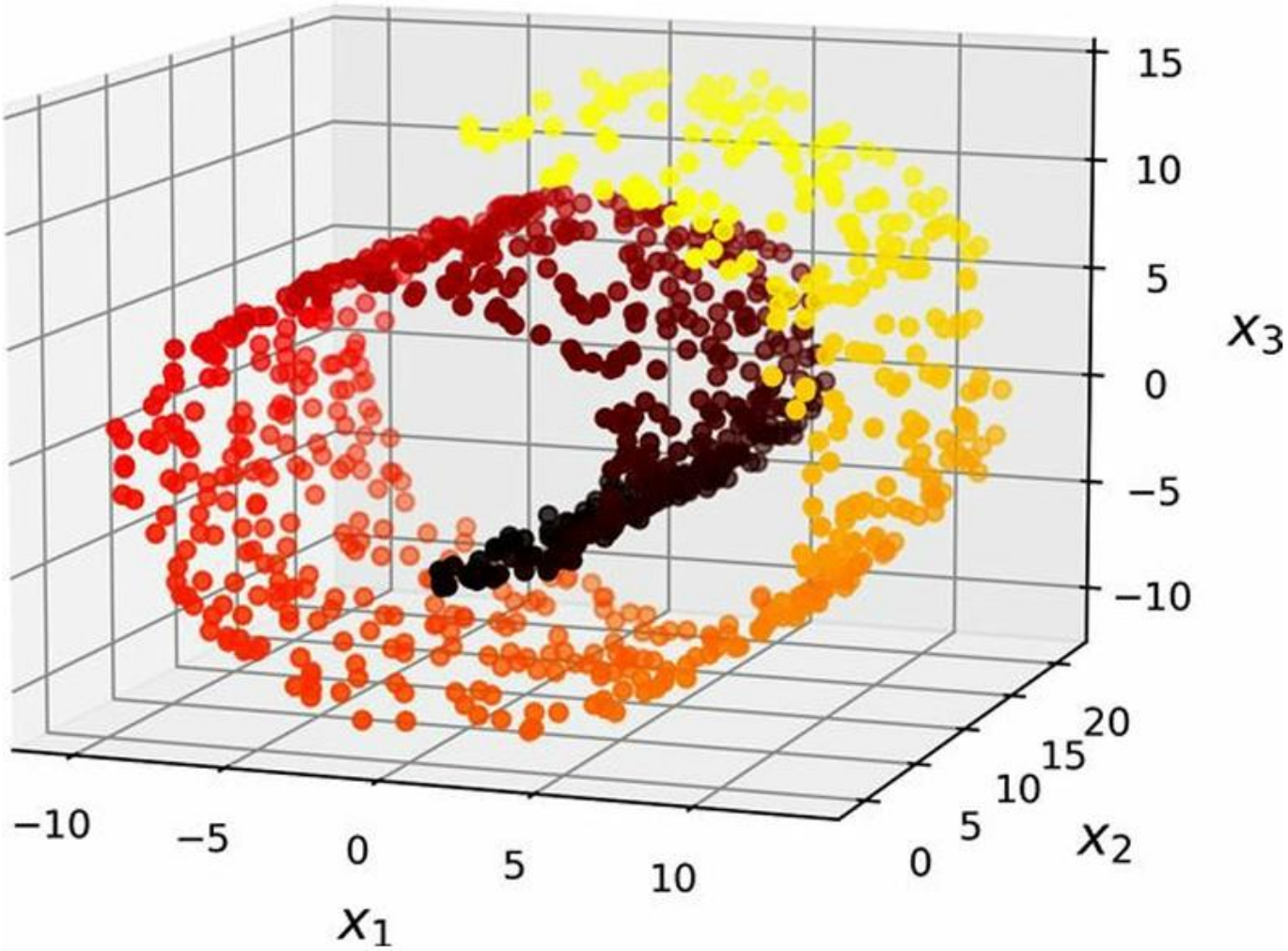


شکل ۸-۳. مجموعه‌داده‌ی دوبعدی جدید پس از پروژکشن

© 2025 Mohsen Zare. All rights reserved.

یادگیری منیفلد

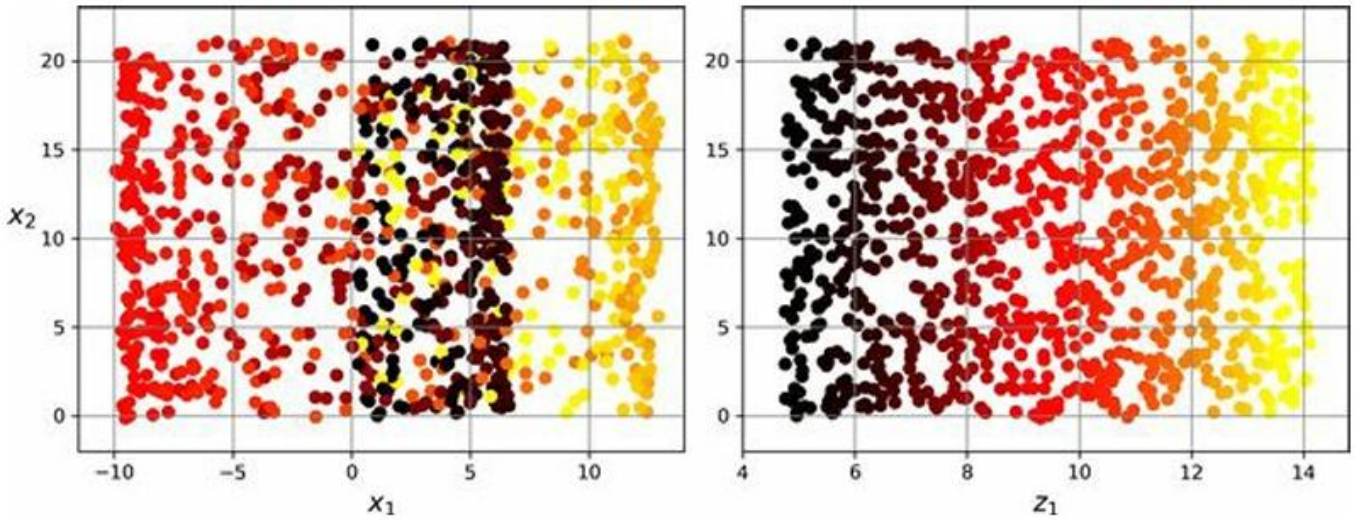
با این حال، تصویرنگاری همیشه بهترین روش برای کاهش ابعاد نیست. در بسیاری از موارد، زیرفضای کم‌بعد ممکن است پیچ و خم داشته باشد، مانند مجموعه‌داده‌ی معروف کوچک، ساده و آزمایشی «رولت سوئیسی(Swiss roll)» که در شکل ۸-۴ نمایش داده شده است.



شکل ۸-۴. مجموعه داده‌های رولت سوئیسی

با تصویر کردنِ ساده روی یک صفحه (مثلاً با حذف مؤلفه‌ی χ_3)، باعث می‌شود لایه‌های مختلف ساختار رولت سوئیسی روی هم فشرده شوند، همان‌طور که در سمت چپ شکل ۸-۵ نشان داده شده است. چیزی که احتمالاً می‌خواهید، باز کردن ساختار رولت سوئیسی است تا به داده‌ای دوبعدی برسید، همان‌طور که در سمت راست شکل ۸-۵ دیده می‌شود.

© 2025 Mohsen Zare. All rights reserved.



شکل ۸-۵. فشرده‌سازی با تصویر کردن روی یک صفحه (سمت چپ) در برابر باز کردن ساختار رول سوئیسی (سمت راست)

رولت سوئیسی یک نمونه از یک «منیفلد دوبعدی (2D *manifold*)» است. به‌بیان ساده، یک منیفلد دوبعدی، یک شکل دوبعدی است که می‌تواند در یک فضای با ابعاد بالاتر خم و پیچ‌خورده باشد. به‌طور کلی‌تر، یک منیفلد با بعد d، بخشی از یک فضای n-بعدی است (که در آن $d < n$) و به صورت محلی شبیه یک ابرصفحه (hyperplane) با بعد d می‌باشد. در مورد ساختار رولت سوئیسی، $d = 2$ و $n = 3$ است: این ساختار به‌صورت محلی شبیه یک صفحه دوبعدی است، اما در بعد سوم پیچیده شده است.

بسیاری از الگوریتم‌های کاهش ابعاد با مدل‌سازی منیفلدی ای کار می‌کنند که داده‌های آموزشی روی آن قرار دارند؛ به این روش «یادگیری منیفلد (*Manifold Learning*)» گفته می‌شود. این روش بر «فرض منیفلد (*manifold assumption*)» که به آن «فرضیه منیفلد (*manifold hypothesis*)» نیز گفته می‌شود، تکیه دارد؛ فرضی که معتقد است بیشتر داده‌های واقعی با ابعاد بالا، نزدیک به یک منیفلد با ابعاد بسیار پایین‌تر قرار دارند. این فرضیه اغلب به‌صورت تجربی مشاهده می‌شود.

یک بار دیگر مجموعه داده‌ی MNIST را تصور کنید: همه تصاویر اعداد دست‌نویس شباهت‌هایی با هم دارند. این تصاویر از خطوط متصل تشکیل شده‌اند، حاشیه‌ها سفید است و به طور تقریبی در مرکز قرار دارند. اگر تصاویر را به‌صورت تصادفی تولید کنید، تنها بخش بسیار ناچیزی از آنها شبیه اعداد دست‌نویس خواهد بود. به عبارت دیگر، درجه آزادی (degrees of freedom) که برای ساختن تصویر یک عدد دست‌نویس دارید، به‌طور چشمگیری کمتر از درجه آزادی‌ای است که اگر اجازه داشته باشید

© 2025 Mohsen Zare. All rights reserved.

هر تصویری را به دلخواه خود بسازید. این محدودیت‌ها باعث می‌شوند که داده‌ها به سمت یک منیفلد با بعد پایین‌تر فشرده شوند.

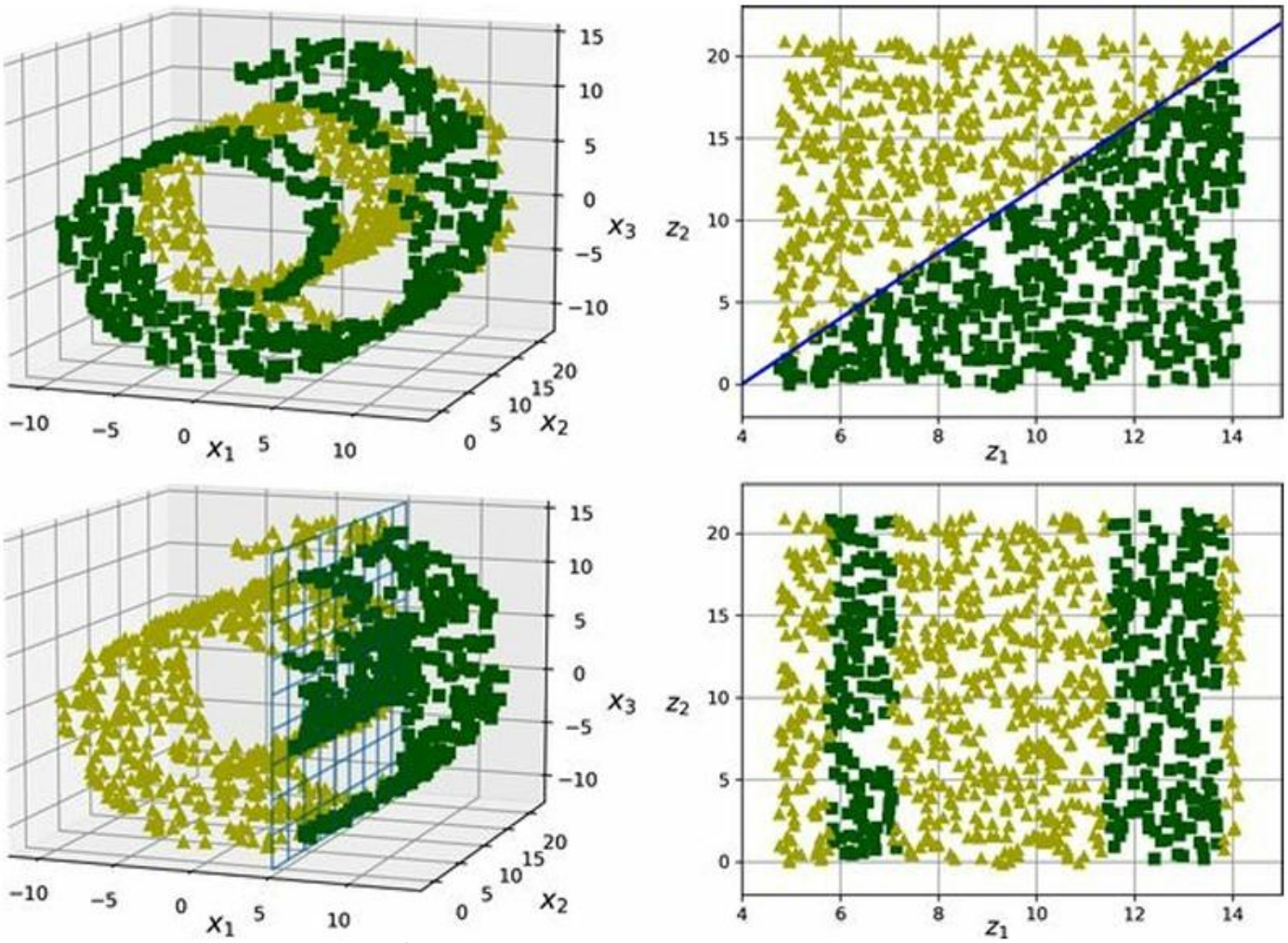
فرض منیفلد معمولاً با یک فرض ضمنی دیگر همراه است: اینکه اگر مسئله مورد نظر (مثلاً طبقه‌بندی یا رگرسیون) در فضای با ابعاد پایین‌تر منیفلد بیان شود، ساده‌تر خواهد بود. برای مثال، در ردیف بالای شکل ۸-۶، رول سوئیسی به دو کلاس تقسیم شده است: در فضای سه‌بعدی (در سمت چپ)، مرز تصمیم‌گیری نسبتاً پیچیده خواهد بود، اما در فضای منیفلدِ بازشده‌ی دوبعدی (در سمت راست)، مرز تصمیم‌گیری یک خط مستقیم است.

بااین‌حال، این فرض ضمنی همیشه صادق نیست. برای مثال، در ردیف پایین شکل ۸-۶، مرز تصمیم‌گیری در محل $x_1 = 5$ قرار دارد. این مرز تصمیم‌گیری در فضای سه‌بعدی اصلی بسیار ساده به نظر می‌رسد (یک صفحه عمودی)، اما در منیفلد بازشده (unrolled manifold) پیچیده‌تر دیده می‌شود: مجموعه‌ای از چهار پاره‌خط مستقل.

به‌طور خلاصه، کاهش ابعاد مجموعه‌ی آموزشی پیش از آموزش یک مدل معمولاً باعث افزایش سرعت آموزش می‌شود، اما لزوماً به راه‌حل بهتر یا ساده‌تری منجر نمی‌شود؛ همه‌چیز به خودِ مجموعه داده بستگی دارد.

امیدوارم اکنون درک خوبی از «نفرین بُعد بالا» یا «مُعضل ابعاد بالا» و نحوه‌ی مقابله‌ی الگوریتم‌های کاهش ابعاد با آن، به‌ویژه در شرایطی که فرض منیفلد برقرار است، به‌دست آورده باشید. در ادامه‌ی این فصل، با برخی از محبوب‌ترین الگوریتم‌های کاهش ابعاد آشنا خواهیم شد.

© 2025 Mohsen Zare. All rights reserved.



شکل ۸-۶. مرز تصمیم ممکن است در ابعاد پایین‌تر لزوماً ساده‌تر نباشد

© 2025 Mohsen Zare. All rights reserved.

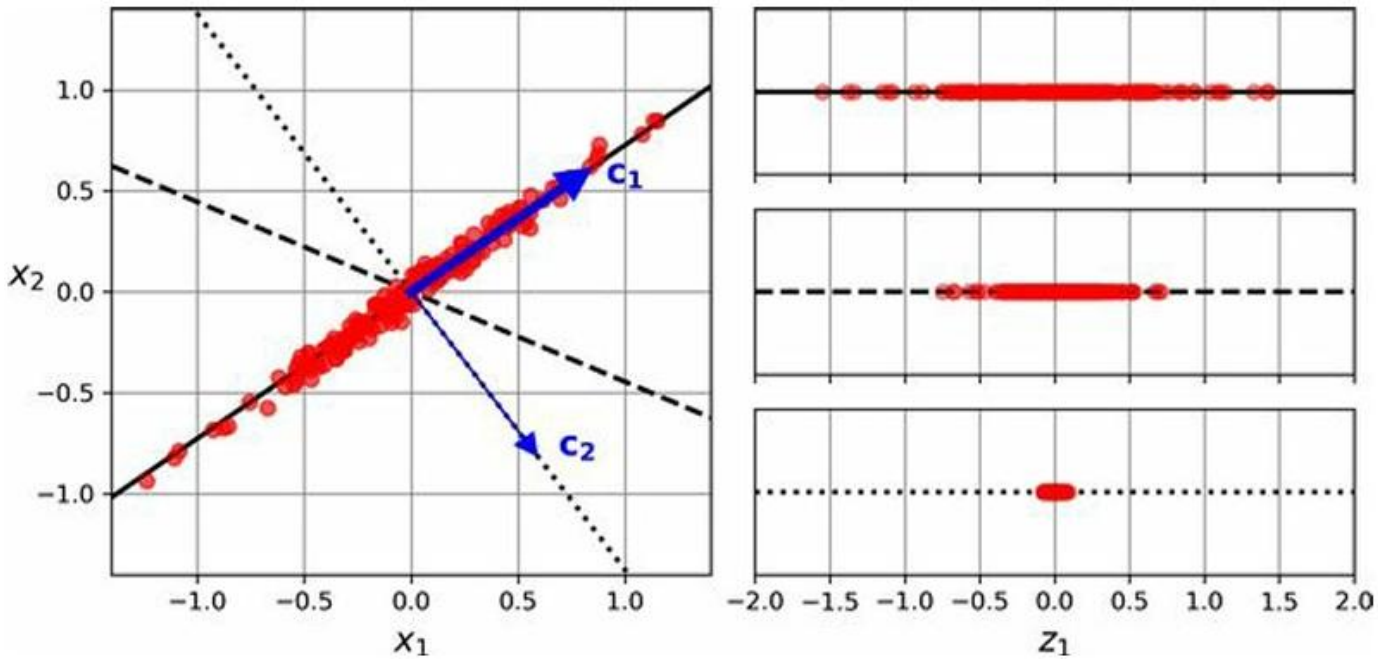
تحلیل مؤلفه‌های اصلی (PCA)

«تحلیل مؤلفه‌های اصلی (PCA: *Principal Component Analysis*)» بی‌شک پرکاربردترین الگوریتم کاهش ابعاد به شمار می‌رود. ابتدا اَبَرصفحه‌ای را که به داده‌ها نزدیک‌ترین است، شناسایی می‌کند، سپس داده‌ها را مانند شکل ۸-۲ بر روی آن ابرصفحه نگاشت (project) می‌کند.

© 2025 Mohsen Zare. All rights reserved.

حفظ واریانس

قبل از اینکه بتوانید مجموعه داده‌ی آموزشی را روی یک ابرصفحه با ابعاد کمتر پروژکت(نگاشت) کنید، ابتدا باید ابرصفحه مناسب را انتخاب کنید. برای مثال، یک مجموعه داده ساده دوبعدی در سمت چپ شکل ۸-۷ نشان داده شده است، همراه با سه محور مختلف (یعنی ابرصفحه‌های یک‌بعدی). در سمت راست، نتیجه پروژکت مجموعه داده روی هر یک از این محورها نمایش داده شده است. همانطور که می‌بینید، پروژکشن روی خطِ پررنگ بیشترین واریانس را حفظ می‌کند (بالا)، در حالی که پروژکشن روی خط نقطه‌چین واریانس بسیار کمی را حفظ می‌کند (پایین) و پروژکشن روی خط‌چین مقداری متوسط از واریانس را حفظ می‌کند (وسط).



شکل ۸-۷. انتخاب زیرفضایی که داده‌ها روی آن تصویر می‌شوند

انتخاب محوری که بیشترین مقدار واریانس را حفظ می‌کند منطقی به نظر می‌رسد، زیرا احتمالاً اطلاعات کمتری نسبت به سایر پروژکشن‌ها از دست می‌دهد. راه دیگری برای توجیه این انتخاب این است که این محور، محوری است که میانگین مربعات فاصله بین مجموعه داده اصلی و پروژکشن آن روی آن محور را به حداقل می‌رساند. این همان ایده نسبتاً ساده‌ای است که پشت PCA قرار دارد.

© 2025 Mohsen Zare. All rights reserved.

مولفه‌های اصلی

PCA محور (یا جهتی) را پیدا می‌کند که بیشترین مقدار واریانس (تغییرات داده‌ها) را در مجموعه‌ی آموزش نشان می‌دهد. در شکل ۸-۷، این محور همان خط پیوسته است. همچنین محور دومی را پیدا می‌کند که بر محور اول عمود است و بیشترین مقدار واریانس باقی‌مانده را توضیح می‌دهد. در این مثال دوبعدی، گزینه‌ای نیست: محور دوم همان خط نقطه‌چین است. اگر داده‌ها در ابعاد بالاتری بودند، PCA محور سومی هم پیدا می‌کرد که عمود بر دو محور قبلی است، سپس محور چهارم، پنجم و... تا هر تعداد محوری که برابر با تعداد ابعاد داده‌ها باشد.

محور i اُم را i اُمین «مؤلفه اصلی داده (*Principal Component* یا PC)» می‌نامند. در شکل ۸-۷، مؤلفه اصلی اول (PC1) همان محوری است که بردار $\mathbf{c}_1$ روی آن قرار دارد و مؤلفه اصلی دوم (PC2) محوری است که بردار $\mathbf{c}_2$ روی آن قرار گرفته است. در شکل ۸-۲، دو مؤلفه اصلی اول (PC1 و PC2) روی صفحه‌ی نگاشت(projection plane) قرار دارند و مؤلفه اصلی سوم (PC3) محوری است که عمود بر آن صفحه است. بعد از پروجکشن (تصویرنگاری داده‌ها روی صفحه‌ی با بُعد پایین)، در شکل ۸-۳، مؤلفه‌ی اصلی اول (PC1) با محور z_1 مطابقت دارد و مؤلفه‌ی اصلی دوم (PC2) با محور z_2 هم‌راستا است.

نکته

برای هر مؤلفه‌ی اصلی، روش PCA یک بردارِ واحدِ میانگین-صفر (zero-centered unit vector) پیدا می‌کند که در راستای همان مؤلفه قرار دارد. از آن‌جا که دو بردار یکه‌ی مخالف، در واقع روی یک محور قرار دارند، جهت این بردارها در خروجی الگوریتم PCA همواره ثابت نیست. به‌عبارت دیگر، اگر مجموعه‌ی آموزش کمی تغییر کند و دوباره PCA اجرا شود، ممکن است بردارهای به‌دست‌آمده در جهتی مخالفِ بردارهای قبلی قرار گیرند. با این حال، این بردارهای واحد معمولاً همچنان روی همان محورها قرار دارند. در برخی موارد، اگر واریانس‌ها در این دو محور بسیار به هم نزدیک باشند، ممکن است این دو بردار بچرخند یا جای یکدیگر را بگیرند، اما صفحه‌ای که این بردارها تعریف می‌کنند معمولاً ثابت باقی می‌ماند.

© 2025 Mohsen Zare. All rights reserved.

پس چگونه می‌توان مؤلفه‌های اصلی یک مجموعه داده آموزشی را پیدا کرد؟ خوشبختانه، یک روش استاندارد تجزیه ماتریسی به نام «تجزیه مقادیر منفرد (SVD: *singular value decomposition*)» وجود دارد که می‌تواند ماتریس مجموعه داده آموزشی $\mathbf{X}$ را به حاصل‌ضرب سه ماتریس $\mathbf{U\,\Sigma\,V^T}$ تجزیه کند، که در آن ماتریس $\mathbf{V}$ شامل بردارهای واحدی است که مؤلفه‌های اصلی مورد نظر را تعریف می‌کنند، همانطور که در رابطه ۸-۱ نشان داده شده است.

معادله ۸-۱. ماتریس مؤلفه‌های اصلی

$$V = \|c_1 c_2 \cdots c_n\|$$

کد پایتون زیر با استفاده از تابع svd() از کتابخانه NumPy، تمام مؤلفه‌های اصلی مجموعه داده‌ی سه‌بعدی نشان‌داده‌شده در شکل ۸-۲ را محاسبه می‌کند، سپس دو بردار واحدی که مؤلفه‌های اصلی اول و دوم را تعریف می‌کنند، استخراج می‌کند:

```
import numpy as np

X = [...] # create a small 3D dataset
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt[0]
c2 = Vt[1]
```

هشدار

PCA فرض می‌کند که داده‌ها حول مبدأ مختصات (مقدار صفر) متمرکز شده‌اند. همان‌طور که خواهید دید، کلاس‌های PCA در کتابخانه‌ی Scikit-Learn به‌طور خودکار داده‌ها را برای شما مرکزسازی(center) می‌کنند. اگر خودتان الگوریتم PCA را پیاده‌سازی می‌کنید (مانند مثالی که در بخش قبل دیدید)، یا اگر از کتابخانه‌های دیگر استفاده می‌کنید، فراموش نکنید که ابتدا داده‌ها را مرکزسازی کنید.

© 2025 Mohsen Zare. All rights reserved.

پروجکت کردن به ابعاد پایین‌تر (d بُعد)

پس از محاسبه مؤلفه‌های اصلی (PCها)، می‌توان داده‌ها را روی زیرفضایی (subspace) با ابعاد پایین‌تر تصویرنگاری(نگاشت یا پروجکت) کرد؛ یعنی آن‌ها را به تعداد کمی از مؤلفه‌های اصلی کاهش داد. این فرآیند باعث کاهش ابعاد (dimensionality reduction) داده‌ها می‌شود، در حالی که بیشترین اطلاعات ممکن را حفظ می‌کند. وقتی تمام مؤلفه‌های اصلی را شناسایی کردید، می‌توانید ابعاد داده‌ها را به d بُعد کاهش دهید، با این کار که داده‌ها را روی اَبَرصفحه‌ای تصویر کنید که توسط d مؤلفه‌ی اصلی اول تعریف شده است. انتخاب این ابرصفحه تضمین می‌کند که بیشترین واریانس ممکن از داده‌ها در فرآیند تصویرسازی (پروجکشن) حفظ شود. برای مثال، در شکل ۸-۲، مجموعه‌داده‌ی سه‌بعدی روی یک صفحه‌ی دوبعدی تصویر شده که توسط دو مؤلفه‌ی اصلی اول تعریف می‌شود، و بخش بزرگی از واریانس داده‌ها را حفظ می‌کند. در نتیجه، تصویر دوبعدی به‌دست‌آمده شباهت زیادی به مجموعه‌داده‌ی سه‌بعدی اصلی دارد.

برای تصویر کردن مجموعه‌ی آموزشی روی ابرصفحه و به‌دست‌آوردن یک مجموعه‌داده‌ی کاهش‌یافته با بُعد d (که آن را با $\mathbf{X}_{d-proj}$ نمایش می‌دهیم)، باید ضرب ماتریسی بین ماتریس مجموعه‌ی آموزشی $\mathbf{X}$ و ماتریس $\mathbf{W}_d$ را محاسبه کنید. ماتریس $\mathbf{W}_d$ شامل اولین d ستون از ماتریس $\mathbf{V}$ است؛ همان‌طور که در معادله‌ی ۸-۲ نشان داده شده است.

معادله ۸-۲. تصویر کردن مجموعه آموزشی به فضای با بُعد d

$$\mathrm{X}_{\mathrm{d-proj}} = \mathrm{X}\,\mathrm{W}_{\mathrm{d}}$$

کد پایتون زیر، مجموعه داده‌ی آموزشی را روی صفحه‌ای که توسط دو مؤلفه‌ی اصلی اول تعریف شده، تصویر می‌کند:

```
W2 = Vt[:2].T
X2D = X_centered @ W2
```

خب! حالا دیگر می‌دانید که چطور می‌توان بُعد هر مجموعه داده‌ای را کاهش داد: با پروژکت کردن داده‌ها روی زیرفضایی با تعداد بُعد دلخواه، و در عین حال حفظ بیشترین میزان از واریانس (اطلاعات)

© 2025 Mohsen Zare. All rights reserved.

موجود در داده‌ها. این همان هدف اصلی تحلیل مؤلفه‌های اصلی (PCA) است: کاهش ابعاد داده‌ها بدون از دست دادن اطلاعات مهم.

© 2025 Mohsen Zare. All rights reserved.

استفاده از Scikit-Learn

کتابخانه‌ی Scikit-Learn ابزارهای بسیار ساده و قدرتمندی برای اجرای PCA در اختیار شما قرار می‌دهد. این ابزارها نه‌تنها عملیات کاهش ابعاد را انجام می‌دهند، بلکه به‌طور خودکار داده‌ها را مرکزسازی‌شده (centered) می‌کنند و مؤلفه‌های اصلی را محاسبه می‌نمایند. کلاس PCA در Scikit-Learn برای پیاده‌سازی تحلیل مؤلفه‌های اصلی (PCA) از تجزیه مقدار منفرد (SVD) استفاده می‌کند، دقیقاً همان روشی که پیش‌تر در این فصل به‌صورت دستی پیاده‌سازی کردیم. کد زیر با استفاده از کلاس PCA در کتابخانه Scikit-Learn، ابعاد داده‌ها را به‌طور خودکار به دو بُعد کاهش می‌دهد. توجه داشته باشید که این کلاس به‌صورت خودکار داده‌ها را در مرکز قرار میدهد، بنابراین نیازی نیست خودتان این کار را انجام دهید:

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X2D = pca.fit_transform(X)
```

پس از آموزش(fitting) ترنسفورمر PCA روی مجموعه داده، ویژگی _components آن شامل ترانهاده‌ی ماتریس $\mathbf{W}_d$ است؛ یعنی برای هر یک از d مؤلفه اصلی اول، یک سطر در آن قرار دارد.

© 2025 Mohsen Zare. All rights reserved.

نسبت واریانس توضیح داده شده

یکی دیگر از اطلاعات مفید، «نسبت واریانس توضیح‌داده‌شده(*explained variance ratio*)» توسط هر مؤلفه اصلی است، که از طریق متغیر explained_variance_ratio_ در دسترس است. این نسبت نشان می‌دهد چه بخشی از واریانس کل داده‌ها در راستای هر مؤلفه اصلی قرار دارد. برای مثال، بیایید نسبت واریانس توضیح‌داده‌شده‌ی دو مؤلفه اصلی اول را در مجموعه‌داده‌ی سه‌بعدی نمایش‌داده‌شده در شکل ۸-۲ بررسی کنیم:

```
>>> pca.explained_variance_ratio_
array([0.7578477 , 0.15186921])
```

این خروجی به ما می‌گوید که حدود ۷۶% از واریانس مجموعه‌داده روی مؤلفه اصلیِ اول قرار دارد، و حدود ۱۵% از واریانس نیز روی مؤلفه اصلی دوم قرار گرفته است. در نتیجه، حدود ۹% از واریانس برای مؤلفه اصلی سوم باقی می‌ماند، بنابراین می‌توان به‌طور منطقی فرض کرد که مؤلفه سوم احتمالاً اطلاعات کمی را در بر دارد.

© 2025 Mohsen Zare. All rights reserved.

انتخاب تعداد مناسب بُعدها

به‌جای انتخاب دلخواه و خودسرانه تعداد بُعدهایی که می‌خواهید کاهش دهید، بهتر است تعداد بُعدهایی را انتخاب کنید که مجموع واریانس توضیح‌داده‌شده‌ی آن‌ها به مقدار کافیِ بزرگی برسد—مثلاً ۹۵%. (البته یک استثنا برای این قاعده وجود دارد: اگر هدف شما از کاهش ابعاد، نمایش یا مصورسازی داده‌ها باشد، در این صورت بهتر است تعداد ابعاد را به ۲ یا ۳ کاهش دهید.)

کد زیر مجموعه‌داده‌ی MNIST (که در فصل ۳ معرفی شد) را بارگذاری و تقسیم می‌کند، سپس PCA را بدون کاهش ابعاد اجرا می‌کند و در ادامه، حداقل تعداد ابعاد لازم برای حفظ ۹۵% از واریانس داده‌های آموزشی را محاسبه می‌کند:

```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', as_frame=False)
X_train, y_train = mnist.data[:60_000], mnist.target[:60_000]
X_test, y_test = mnist.data[60_000:], mnist.target[60_000:]

pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1 # d equals 154
```

سپس می‌توانید مقدار n_components را برابر با d قرار دهید و دوباره PCA را اجرا کنید، اما گزینه‌ی بهتری وجود دارد. به جای اینکه تعداد مؤلفه‌های اصلی مورد نظر برای حفظ را مشخص کنید، می‌توانید مقدار n_components را به صورت عددی اعشاری بین ۰٫۰ تا ۱٫۰ تنظیم کنید که نشان‌دهنده نسبت واریانس مورد نظر برای حفظ باشد:

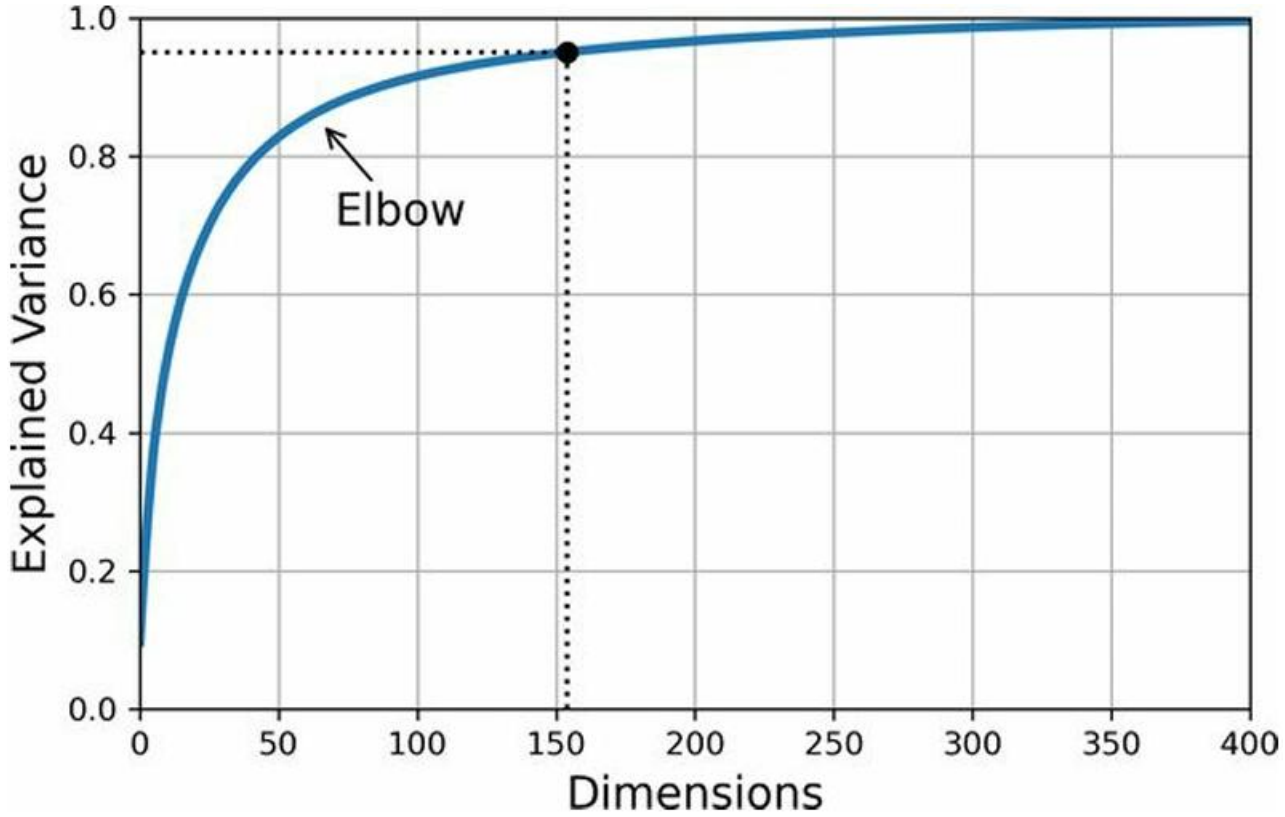
```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

تعداد واقعی مؤلفه‌ها در هنگام آموزش تعیین می‌شود و در ویژگی n_components_ ذخیره می‌گردد:

```
>>> pca.n_components_
```

© 2025 Mohsen Zare. All rights reserved.

گزینه‌ی دیگری که وجود دارد، رسم نسبت واریانس توضیح داده شده به ازای تعداد ابعاد است (کافی است مجموع تجمعی(cumsum) را رسم کنید؛ همانند شکل ۸-۸). معمولاً در این نمودار یک نقطه‌ی زانویی (Elbow) دیده می‌شود؛ جایی که رشد سریع واریانس توضیح داده شده متوقف می‌شود. در این حالت می‌توان دید که کاهش ابعاد داده‌ها تا حدود ۱۰۰ بُعد، بخش زیادی از واریانس توضیح داده شده را از بین نمی‌برد و همچنان مقدار قابل‌توجهی از اطلاعات حفظ می‌شود.



شکل ۸-۸. واریانس توضیح‌داده‌شده به‌عنوان تابعی از تعداد ابعاد

در نهایت، اگر از کاهش ابعاد به‌عنوان یک گام پیش‌پردازشی برای یک مسئله‌ی یادگیری نظارت‌شده (مثلاً طبقه‌بندی) استفاده می‌کنید، می‌توانید تعداد ابعاد را مانند سایر هایپرپارامترها تنظیم (tune) کنید (نگاه کنید به فصل ۲). برای مثال، قطعه‌کد زیر یک پایپ‌لاین(مسیر پردازش داده) دو مرحله‌ای ایجاد می‌کند: ابتدا با استفاده از PCA ابعاد داده‌ها کاهش پیدا می‌کند، و سپس یک طبقه‌بند جنگل تصادفی روی داده‌های کاهش‌یافته اعمال می‌شود. در ادامه، از ابزار RandomizedSearchCV استفاده می‌شود تا یک ترکیب مناسب از هایپرپارامترها برای هر دو مرحله یعنی PCA و طبقه‌بند جنگل تصادفی پیدا شود. این مثال یک جست‌وجوی سریع انجام می‌دهد: تنها دو هایپرپارامتر را تنظیم

© 2025 Mohsen Zare. All rights reserved.

می‌کند، آموزش را فقط روی ۱,۰۰۰ نمونه انجام می‌دهد و جست‌وجو را در ۱۰ تکرار اجرا می‌کند. اما اگر زمان کافی در اختیار دارید، می‌توانید یک جست‌وجوی دقیق‌تر و گسترده‌تر انجام دهید.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import make_pipeline

clf = make_pipeline(PCA(random_state=42),
        RandomForestClassifier(random_state=42))
param_distrib = {
  "pca__n_components": np.arange(10, 80),
  "randomforestclassifier__n_estimators": np.arange(50, 500)
}
rnd_search = RandomizedSearchCV(clf, param_distrib, n_iter=10, cv=3,
             random_state=42)
rnd_search.fit(X_train[:1000], y_train[:1000])
```

بیایید نگاهی بیندازیم به بهترین مقادیر هایپرپارامترهایی که پیدا شده‌اند:

```
>>> print(rnd_search.best_params_)
{'randomforestclassifier__n_estimators': 465, 'pca__n_components': 23}
```

جالب است که ببینیم تعداد بهینه‌ی مؤلفه‌ها چقدر کم است: ما یک مجموعه داده‌ی ۷۸۴ بعدی را به تنها ۲۳ بعد کاهش دادیم! این موضوع به این دلیل است که از مدل جنگل تصادفی استفاده کردیم که مدل نسبتاً قدرتمندی است. اگر به جای آن از یک مدل طبقه‌بند خطی مثل SGDClassifier استفاده کنیم، جستجو نشان می‌دهد که باید ابعاد بیشتری را حفظ کنیم (حدود ۷۰ بعد).

© 2025 Mohsen Zare. All rights reserved.

فشرده‌سازی با PCA

بعد از کاهش ابعاد، مجموعه داده‌ی آموزشی فضای بسیار کمتری اشغال می‌کند. برای مثال، پس از اعمال PCA روی مجموعه داده MNIST و حفظ ۹۵% از واریانس آن، تعداد ویژگی‌ها به ۱۵۴ کاهش می‌یابد، در حالی که تعداد ویژگی‌های اصلی ۷۸۴ بود. بنابراین اندازه مجموعه داده اکنون کمتر از ۲۰% اندازه اصلی است و تنها ۵% از واریانس آن را از دست داده‌ایم! این نسبت فشرده‌سازی معقولی است و به‌راحتی می‌توان فهمید که چنین کاهش حجمی چقدر می‌تواند سرعت اجرای یک الگوریتم طبقه‌بندی را به طور چشمگیری افزایش دهد.

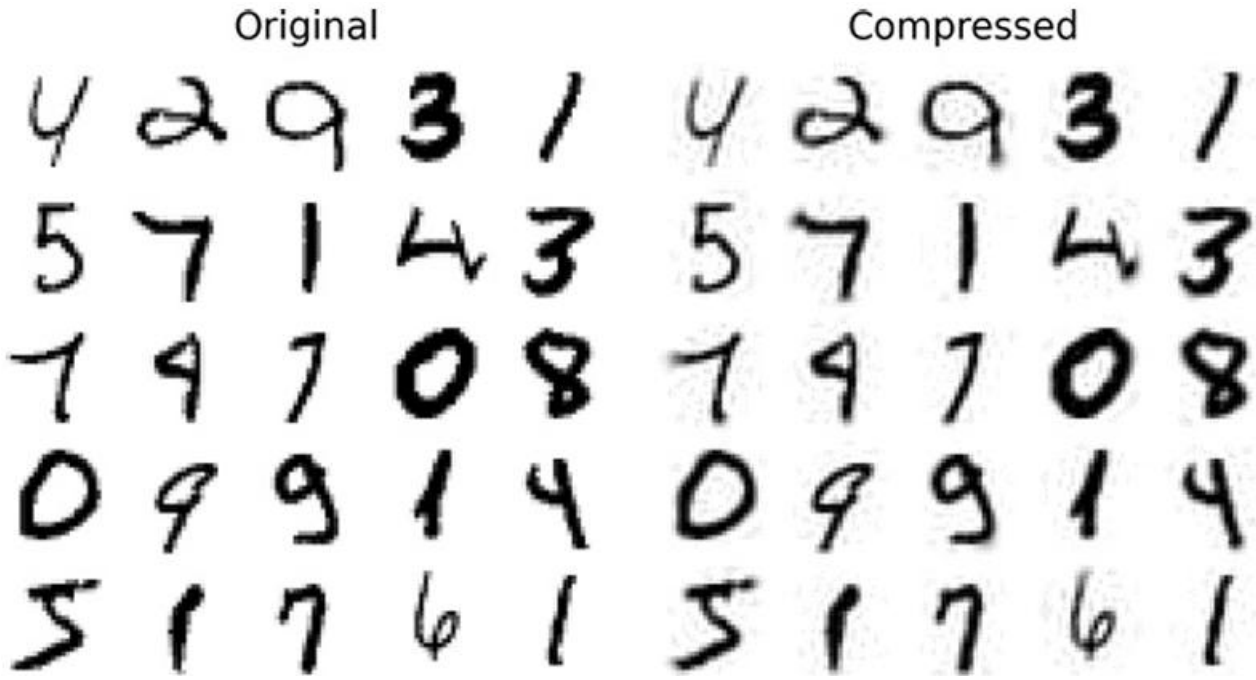
همچنین می‌توان مجموعه داده کاهش‌یافته را دوباره به فضای ۷۸۴ بُعدی اولیه بازگرداند، با اعمال تبدیل معکوسِ تصویرِ پروژکت شده‌ی PCA. با این‌که این کار داده‌های اصلی را دقیقاً بازنمی‌گرداند—چرا که در فرایند فشرده‌سازی مقداری از اطلاعات (در حد همان ۵% واریانس حذف‌شده) از دست رفته است—اما داده‌های بازسازی‌شده معمولاً بسیار نزدیک به داده‌های اولیه خواهند بود. فاصله‌ی میانگینِ مربعیِ بین داده‌های اصلی و داده‌های بازسازی‌شده (یعنی داده‌هایی که ابتدا فشرده و سپس بازسازی شده‌اند)، «خطای بازسازی(*reconstruction error*)» نامیده می‌شود.

متد inverse_transform() این امکان را به ما می‌دهد که مجموعه داده‌ی MNIST کاهش‌یافته را دوباره به فضای اولیه‌ی ۷۸۴ بعدی بازگردانیم. این بازسازی البته دقیقاً با داده‌های اولیه یکسان نخواهد بود، اما تقریب نسبتاً خوبی از آن‌ها ارائه می‌دهد.

```
X_recovered = pca.inverse_transform(X_reduced)
```

شکل ۸-۹ چند نمونه از «تصاویر اعداد» را در مجموعه داده‌ی اولیه‌ی آموزش (در سمت چپ) را نشان می‌دهد، و در کنار آن‌ها نسخه‌های بازسازی‌شده‌ی همان تصاویر را پس از فشرده‌سازی و سپس بازسازی نمایش می‌دهد. می‌توانید ببینید که کیفیت تصاویر کمی کاهش یافته، اما شکل کلی ارقام همچنان به‌خوبی حفظ شده است.

© 2025 Mohsen Zare. All rights reserved.



شکل ۸-۹. فشرده‌سازی مجموعه داده MNIST با حفظ ۹۵% از واریانس داده‌ها

معادله تبدیل معکوس در رابطه‌ی ۸-۳ نشان داده شده است.

معادله ۸-۳. تبدیل معکوس PCA برای بازگرداندن داده‌ها به تعداد ابعاد اولیه

$$X_{recovered} = X_{d-proj} (W_d)^T$$

© 2025 Mohsen Zare. All rights reserved.

PCA تصادفی

اگر هایپرپارامتر svd_solver را روی مقدار "randomized" تنظیم کنید، کتابخانه Scikit-Learn از الگوریتم تصادفی‌ای به نام «PCA تصادفی ($randomized\ PCA$)» استفاده می‌کند که به سرعت تقریب مؤلفه‌های اصلی اول تا d ام را پیدا می‌کند. پیچیدگی محاسباتی آن برابر است با $O(m \times d^2) + O(d^3)$، در حالی که روش کامل SVD دارای پیچیدگی $O(m \times n^2) + O(n^3)$ است؛ بنابراین وقتی d بسیار کوچکتر از n باشد، این روش به‌طور قابل توجهی سریع‌تر از SVD کامل عمل می‌کند:

```
rnd_pca = PCA(n_components=154, svd_solver="randomized", random_state=42)
X_reduced = rnd_pca.fit_transform(X_train)
```

نکته

به طور پیش‌فرض، مقدار svd_solver روی "auto" تنظیم شده است: کتابخانه Scikit-Learn به طور خودکار الگوریتم randomized PCA را استفاده می‌کند اگر ماکزیمم m یا n بزرگ‌تر از ۵۰۰ باشد و n_components عددی صحیح کمتر از ۸۰% از مینیمم m و n باشد؛ در غیر این صورت، روش کامل SVD را به کار می‌گیرد. پس کد قبلی حتی اگر آرگومان svd_solver="randomized" را حذف کنید هم از الگوریتم randomized PCA استفاده خواهد کرد، چون $154 < 0.8 \times 784$ است. اگر بخواهید Scikit-Learn را مجبور کنید که برای نتیجه‌ای دقیق‌تر از روش کامل تجزیه مقدار منفرد (full SVD) استفاده کند، می‌توانید مقدار پارامتر svd_solver را روی "full" تنظیم کنید.

© 2025 Mohsen Zare. All rights reserved.

PCA تدریجی(افزایشی)

یکی از مشکلات نسخه‌های قبلی PCA این است که برای اجرا شدن، نیاز دارند کل داده‌های آموزشی به طور کامل در حافظه قرار داشته باشند. خوشبختانه الگوریتم‌های «PCA افزایشی یا تدریجی (*Incremental PCA* یا IPCA)» توسعه یافته‌اند که به شما اجازه می‌دهند داده‌های آموزشی را به دسته‌های کوچک‌تر (mini-batches) تقسیم کنید و هر بار فقط یک دسته کوچک را به الگوریتم بدهید. این روش برای مجموعه‌داده‌های بزرگ بسیار مفید است و همچنین زمانی که بخواهید PCA را به‌صورت آنلاین (یعنی به‌صورت لحظه‌ای، همزمان با ورود داده‌های جدید) اعمال کنید، کاربرد دارد.

کد زیر مجموعه‌ی آموزش MNIST را به ۱۰۰ مینی‌دسته تقسیم می‌کند (با استفاده از تابع array_split() از کتابخانه‌ی NumPy) و آن‌ها را به کلاس IncrementalPCA در Scikit-Learn می‌دهد تا ابعاد داده‌های MNIST را مانند قبل به ۱۵۴ بعد کاهش دهد. توجه داشته باشید که باید از متد partial_fit() برای هر مینی دسته ای استفاده کنید، نه از متد fit() برای کل مجموعه‌ی آموزش. این موضوع به این دلیل است که در Incremental PCA، داده‌ها به‌صورت مرحله‌ای پردازش می‌شوند و متد partial_fit() اجازه می‌دهد تا مدل به‌تدریج با هر بخش از داده‌ها آموزش ببیند، بدون نیاز به بارگذاری تمام داده‌ها در حافظه.

```
from sklearn.decomposition import IncrementalPCA

n_batches = 100
inc_pca = IncrementalPCA(n_components=154)
for X_batch in np.array_split(X_train, n_batches):
  inc_pca.partial_fit(X_batch)

X_reduced = inc_pca.transform(X_train)
```

به‌صورت جایگزین، می‌توانید از کلاس memmap در کتابخانه‌ی NumPy استفاده کنید. این کلاس به شما اجازه می‌دهد که یک آرایه‌ی بزرگ ذخیره‌شده در یک فایل باینری روی دیسک را طوری مدیریت کنید که انگار به‌طور کامل در حافظه قرار دارد؛ اما در واقع فقط بخش‌هایی از داده که در هر لحظه نیاز است، به حافظه بارگذاری می‌شوند. برای نمایش این موضوع، ابتدا یک فایل حافظه‌نگاشت‌شده

© 2025 Mohsen Zare. All rights reserved.

(memory-mapped یا memmap) ایجاد می‌کنیم و مجموعه‌داده‌ی آموزشی MNIST را در آن کپی می‌کنیم. سپس، با فراخوانی متد flush() اطمینان حاصل می‌کنیم که هر داده‌ای که هنوز در حافظه‌ی نهان (cache) باقی مانده، روی دیسک ذخیره شود. در دنیای واقعی، معمولاً X_train آن‌قدر بزرگ است که در حافظه جا نمی‌گیرد، بنابراین شما آن را به‌صورت تکه‌تکه یا بخش‌بخش (chunk by chunk) بارگذاری می‌کنید و هر بخش را در قسمت مناسب از آرایه‌ی memmap ذخیره می‌کنید:

```
filename = "my_mnist.mmap"
X_mmap = np.memmap(filename, dtype='float32', mode='write', shape=X_train.shape)
X_mmap[:] = X_train # could be a loop instead, saving the data chunk by chunk
X_mmap.flush()
```

سپس می‌توانیم فایل memmap را بارگذاری کرده و از آن درست مانند یک آرایه‌ی معمولی NumPy استفاده کنیم. بیایید از کلاس IncrementalPCA برای کاهش ابعاد آن استفاده کنیم. از آنجایی که این الگوریتم در هر لحظه فقط بخش کوچکی از آرایه را استفاده می‌کند، میزان مصرف حافظه تحت کنترل باقی می‌ماند. این موضوع باعث می‌شود بتوانید به‌جای استفاده از متد partial_fit()، از همان متد معمول fit() استفاده کنید، که کار را بسیار راحت‌تر می‌کند.

```
X_mmap = np.memmap(filename, dtype="float32", mode="readonly").reshape(-1, 784)
batch_size = X_mmap.shape[0] // n_batches
inc_pca = IncrementalPCA(n_components=154, batch_size=batch_size)
inc_pca.fit(X_mmap)
```

هشدار

تنها داده‌های خام باینری در دیسک ذخیره می‌شوند، بنابراین هنگام بارگذاری فایل، باید نوع داده (data type) و شکل آرایه (shape) را مشخص کنید. اگر شکل را مشخص نکنید، تابع np.memmap() یک آرایه یک‌بعدی (1D array) بازمی‌گرداند.

برای مجموعه داده‌های با بُعد بسیار بالا (high-dimensional)، الگوریتم PCA ممکن است بیش

© 2025 Mohsen Zare. All rights reserved.

از حد کند باشد. همان‌طور که قبلاً دیدید، حتی اگر از PCA تصادفی استفاده کنید، پیچیدگی محاسباتی آن همچنان برابر است با $O(m \times d^2) + O(d^3)$، بنابراین تعداد ابعاد هدف d نباید خیلی بزرگ باشد. اگر با مجموعه‌داده‌ای سروکار دارید که ده‌ها هزار ویژگی یا بیشتر دارد (مثل تصاویر)، آموزش مدل ممکن است خیلی کند شود؛ در این حالت بهتر است از روش «تصویرنگاری تصادفی» استفاده کنید.

© 2025 Mohsen Zare. All rights reserved.

تصویرنگاری تصادفی

همان‌طور که از نامش پیداست، الگوریتم تصویرنگاری تصادفی (random projection)، داده‌ها را با استفاده از یک تصویرنگاریِ خطیِ تصادفی به فضای با ابعاد کمتر پروجکت(بازتاب) می‌دهد. شاید این ایده عجیب به نظر برسد، اما واقعیت این است که چنین تصویرنگاری تصادفی ای به احتمال زیاد فاصله‌ها را به خوبی حفظ می‌کند، همان‌طور که «ویلیام بی» به صورت ریاضی اثبات کرده است. ویلیام بی. جانسون و یورام لیندن‌اشتراوس این موضوع را در یک لم (lemma) معروف به اثبات رساندند. بنابراین، دو نمونه‌ی مشابه پس از تصویرنگاری همچنان مشابه باقی می‌مانند و دو نمونه‌ی بسیار متفاوت نیز همچنان بسیار متفاوت خواهند بود.

بدیهی است که هرچه ابعاد بیشتری را حذف کنید، اطلاعات بیشتری از دست می‌رود و فاصله‌ها بیشتر تغییر می‌کنند. پس چطور می‌توان تعداد بهینه‌ی ابعاد را انتخاب کرد؟ خب، جانسون و لیندنشتراس یک معادله ارائه کردند که حداقل تعداد ابعادی را مشخص می‌کند که باید حفظ شوند تا با احتمال بالا تضمین شود فاصله‌ها بیش از حد مجاز(یک تُلِرانس مشخص) تغییر نکنند. برای مثال، اگر یک مجموعه داده داشته باشید که شامل $m = 5{,}000$ نمونه با $n = 20{,}000$ ویژگی برای هر نمونه باشد، و نخواهید فاصله مربعی بین هر دو نمونه بیش از $\varepsilon = 10$ % تغییر کند، باید داده‌ها را به d بعد کاهش دهید که در آن $d \geq 4\, log(m) \,/\, (½\, \varepsilon^2 - ⅓\, \varepsilon^3)$ است؛ که در اینجا برابر با ۷,۳۰۰ بعد می‌شود. این کاهش ابعاد کاملاً چشمگیر است! دقت کنید که در این معادله از n (تعداد ویژگی‌ها) استفاده نمی‌شود، بلکه فقط به m (تعداد نمونه‌ها) و ε (میزان تحمل تغییر فاصله) وابسته است. این معادله توسط تابع johnson_lindenstrauss_min_dim() پیاده‌سازی شده است:

```
>>> from sklearn.random_projection import johnson_lindenstrauss_min_dim
>>> m, ε = 5_000, 0.1
>>> d = johnson_lindenstrauss_min_dim(m, eps=ε)
>>> d
7300
```

حالا می‌توانیم فقط یک ماتریس تصادفی $\mathbf{P}$ با شکل $[d, n]$ تولید کنیم، به‌طوری که هر عنصر آن به‌صورت تصادفی از یک توزیع گاوسی با میانگین 0 و واریانس $1/d$ نمونه‌گیری شده باشد، و از آن برای پروجکشنِ یک مجموعه‌داده از n بُعد به d بُعد استفاده کنیم:

© 2025 Mohsen Zare. All rights reserved.

```
n = 20_000
np.random.seed(42)
P = np.random.randn(d, n) / np.sqrt(d) # std dev = square root of variance
X = np.random.randn(m, n) # generate a fake dataset
X_reduced = X @ P.T
```

همه‌اش همین بود! این روش ساده و کارآمد است و نیازی به آموزش ندارد: تنها چیزی که الگوریتم برای ساخت ماتریس تصادفی نیاز دارد، ابعاد (shape) مجموعه‌داده است؛ خود داده‌ها اصلاً مورد استفاده قرار نمی‌گیرند.

کتابخانه Scikit-Learn کلاسی به نام GaussianRandomProjection ارائه می‌دهد که دقیقاً همین کاری را انجام می‌دهد که توضیح دادیم: وقتی متد fit() را روی آن فراخوانی می‌کنید، ابتدا از تابع johnson_lindenstrauss_min_dim() برای تعیین تعداد ابعاد خروجی استفاده می‌کند، سپس یک ماتریس تصادفی تولید می‌کند و آن را در ویژگی components_ ذخیره می‌کند. سپس، زمانی که متد transform() را فراخوانی می‌کنید، از همین ماتریس تصادفی برای انجام تصویرسازی استفاده می‌شود. هنگام ساختن ترنسفورمر (مانند GaussianRandomProjection)، می‌توانید پارامتر eps را تعیین کنید تا مقدار ε را تنظیم کنید (به‌صورت پیش‌فرض ۰/۱ است)، و همچنین می‌توانید پارامتر n_components را مشخص کنید تا بُعد هدف دلخواه خود (d) را به‌صورت دستی تعیین کنید. مثال کد زیر دقیقاً همان نتیجه‌ای را می‌دهد که کد قبلی تولید می‌کرد (می‌توانید بررسی کنید که gaussian_rnd_proj.components_ با ماتریس تصادفی P برابر است):

```
from sklearn.random_projection import GaussianRandomProjection

gaussian_rnd_proj = GaussianRandomProjection(eps=ε, random_state=42)
X_reduced = gaussian_rnd_proj.fit_transform(X) # same result as above
```

کتابخانه Scikit-Learn همچنین یک ترنسفورمر دیگر برای پروجکشن تصادفی ارائه می‌دهد که به نام SparseRandomProjection شناخته می‌شود. این ترنسفورمر نیز به همان روش، تعداد ابعاد هدف را تعیین می‌کند، یک ماتریس تصادفی با همان ابعاد تولید می‌کند و پروجکشن را به همان شکل انجام می‌دهد. تفاوت اصلی این است که ماتریس تصادفی در این روش تُنُک یا پراکنده (sparse) است. این یعنی حافظه بسیار کمتری مصرف می‌کند: حدود ۲۵ مگابایت به جای تقریباً ۱/۲ گیگابایت در مثال قبلی! همچنین این روش در تولید ماتریس تصادفی و کاهش ابعاد بسیار سریع‌تر است؛ در

© 2025 Mohsen Zare. All rights reserved.

این مثال حدود ۵۰% سریع‌تر عمل می‌کند. علاوه بر این، اگر داده‌های ورودی پراکنده باشند، این تبدیل هم آن‌ها را پراکنده نگه می‌دارد (مگر اینکه گزینه dense_output=True را فعال کنید). در نهایت، این روش هم مانند روش قبلی خاصیت حفظ فاصله‌ها را دارد و کیفیت کاهش ابعاد آن تقریباً مشابه است. به طور خلاصه، معمولاً بهتر است به جای روش اول از این ترنسفورمر استفاده کنید، به‌ویژه وقتی با داده‌های بزرگ یا پراکنده سروکار دارید.

نسبت r بین تعداد عناصر غیرصفر به کل عناصر در ماتریس تصادفی تُنُک را «چگالی($density$)» آن می‌نامند. به‌طور پیش‌فرض، این مقدار برابر با $1/n$ است، که در آن n تعداد ویژگی‌های (ویژگی‌های اصلی) داده‌ی ورودی است. با داشتن ۲۰,۰۰۰ ویژگی، این یعنی تنها حدود ۱ از هر ۱۴۱ سلول در ماتریس تصادفی مقدار غیرصفر دارد؛ این مقدار واقعاً تُنُک است! می‌توانید هایپرپارامتر density را به مقدار دلخواه دیگری تنظیم کنید، اگر ترجیح می‌دهید تراکم ماتریس تصادفی بیشتر یا کمتر باشد. هر سلول در ماتریس تصادفی تُنُک با احتمال r مقدار متفاوت از صفر خواهد داشت. این مقدار غیر صفر به‌صورت تصادفی برابر با $+v$ یا $-v$ (با احتمال مساوی) انتخاب می‌شود، که در آن: $v = 1/dr$. در نتیجه، بیشتر سلول‌های ماتریس صفر هستند، اما سلول‌های غیر صفر دارای مقادیری با اندازه‌ی ثابت هستند که به‌صورت تصادفی مثبت یا منفی می‌باشند. این ویژگی باعث کاهش شدید مصرف حافظه و افزایش سرعت پردازش می‌شود، بدون آنکه خاصیت حفظ فاصله‌ها (distance-preserving) از بین برود.

اگر بخواهید تبدیل معکوس را انجام دهید، ابتدا باید شبه‌معکوس (pseudo-inverse) ماتریس مؤلفه‌ها را با استفاده از تابع pinv() در کتابخانه‌ی SciPy محاسبه کرده، و سپس داده‌های کاهش‌یافته را در ترانهاده‌ی این شبه‌معکوس ضرب کنید:

```
components_pinv = np.linalg.pinv(gaussian_rnd_proj.components_)
X_recovered = X_reduced @ components_pinv.T
```

© 2025 Mohsen Zare. All rights reserved.

هشدار

محاسبه‌ی شبه‌معکوس می‌تواند زمان زیادی ببرد، به‌ویژه اگر ماتریس مؤلفه‌ها بزرگ باشد، چراکه پیچیدگی محاسباتی تابع pinv() وقتیکه $d < n$ باشد برابر است با $O(dn^2)$ ، و یا در غیر این صورت برابر با $O(nd^2)$.

خلاصه اینکه، تصویرنگاری تصادفی (Random Projection) یک الگوریتم کاهش ابعاد ساده، سریع، کم‌مصرف از نظر حافظه، و به‌طرز شگفت‌آوری قدرتمند است که باید به‌ویژه هنگام کار با مجموعه‌داده‌های با ابعاد بالا، در نظر داشته باشید.

نکته

تصویرنگاری تصادفی همیشه برای کاهش ابعاد مجموعه‌داده‌های بزرگ به‌کار نمی‌رود. برای مثال، مقاله‌ای منتشرشده در سال ۲۰۱۷ توسط Sanjoy Dasgupta و همکاران... نشان داد که مغز مگس میوه (fruit fly) نوعی مشابه از تصویرنگاری تصادفی را پیاده‌سازی می‌کند تا ورودی‌های بویایی (olfactory inputs) که متراکم و کم‌بعد هستند را به خروجی‌هایی پراکنده، پُربُعد و دودویی (binary) تصویر کند: برای هر بو، تنها بخش کوچکی از نورون‌های خروجی فعال می‌شوند، اما بوهای مشابه، تعداد زیادی از همان نورون‌ها را فعال می‌کنند. این سازوکار شبیه یک الگوریتم شناخته‌شده به نام «هشینگِ حساس به همجواری (*Locality Sensitive Hashing* یا به اختصار LSH)» است، که معمولاً در موتورهای جستجو برای گروه‌بندی اسناد مشابه مورد استفاده قرار می‌گیرد. Hashing به معنی نگاشت داده‌ها به کدهای عددی یا آدرس در جدول برای دسترسی سریع و Locality Sensitive Hashing الگویی برای نگاشت داده‌ها به فضای پایین‌بعدی به طوری که نمونه‌های نزدیک در فضای اصلی، با احتمال زیاد به همان «سطل هش» بروند؛ استفاده در Approximate Nearest Neighbor Search.

© 2025 Mohsen Zare. All rights reserved.

الگوریتم LLE

«نمایش نَهُفته یا بردار نَهُفته خطی محلی (LLE: *Locally Linear Embedding*)» یک تکنیک «کاهش ابعاد غیرخطی (*Nonlinear Dimensionality Reduction* یا NLDR)» است. این روش نوعی تکنیک یادگیری مَنیفُلدی است که برخلاف روش‌هایی مانند PCA و تصویرنگاری تصادفی، مبتنی بر نگاشت یا پروجکشن نیست. به‌طور خلاصه، LLE ابتدا بررسی می‌کند که هر نمونه‌ی آموزشی چگونه به‌صورت خطی با نزدیک‌ترین همسایگانش مرتبط است، و سپس به‌دنبال یک نمایش کم‌بعد از داده‌ها می‌گردد که در آن، این روابط محلی تا حد امکان حفظ شده باشند (جزئیات بیشتر در ادامه خواهد آمد). این رویکرد آن را به گزینه‌ای بسیار مناسب برای باز کردن منیفولدهای پیچ‌خورده تبدیل می‌کند، به‌ویژه زمانی که داده‌ها نویز زیادی نداشته باشند.

کد زیر یک رولت سوئیسی تولید می‌کند و سپس با استفاده از کلاس LocallyLinearEmbedding از کتابخانه Scikit-Learn آن را باز می‌کند (unrolling):

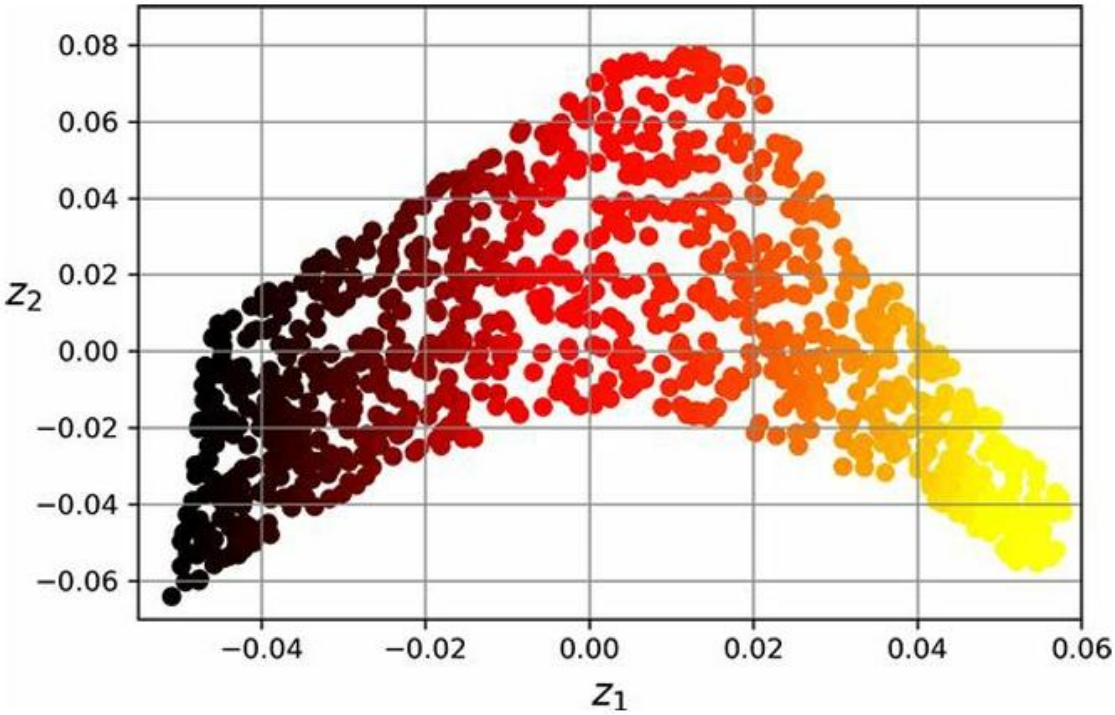
```
from sklearn.datasets import make_swiss_roll
from sklearn.manifold import LocallyLinearEmbedding

X_swiss, t = make_swiss_roll(n_samples=1000, noise=0.2, random_state=42)
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10, random_state=42)
X_unrolled = lle.fit_transform(X_swiss)
```

متغیر "t" یک آرایه‌ی یک‌بعدی از نوع NumPy است که موقعیت هر نمونه را در امتداد محور پیچ‌خورده‌ی رولت سوئیسی نشان می‌دهد. ما از این متغیر در این مثال استفاده نمی‌کنیم، اما می‌توان از آن به‌عنوان هدف (target) در یک مسئله‌ی رگرسیون غیرخطی استفاده کرد.

داده‌ی دوبعدی حاصل در شکل ۸-۱۰ نشان داده شده است. همان‌طور که می‌بینید، رولت سوئیسی به‌طور کامل باز شده و فواصل بین نمونه‌ها به‌خوبی در مقیاس محلی حفظ شده‌اند. با این حال، فواصل در مقیاس بزرگ‌تر حفظ نشده‌اند: رولت بازشده‌ی سوئیسی باید به شکل یک مستطیل باشد،

© 2025 Mohsen Zare. All rights reserved.

نه نواری کشیده و پیچ‌خورده مانند آنچه در تصویر می‌بینید. با این حال، الگوریتم LLE عملکرد نسبتاً خوبی در مدل‌سازی منیفلد از خود نشان داده است.



شکل ۸-۱۰. باز شدن رولت سوئیسی با استفاده از الگوریتم LLE

در اینجا نحوه کار LLE آمده است: برای هر نمونه‌ی آموزشی $\mathbf{x}^{(i)}$، الگوریتم ابتدا k-همسایه‌ی نزدیک آن را شناسایی می‌کند (در کد بالا $k = 10$ است)، سپس تلاش می‌کند تا $\mathbf{x}^{(i)}$ را به‌صورت ترکیبی خطی از این همسایگان بازسازی کند. به‌طور دقیق‌تر، الگوریتم تلاش می‌کند وزن‌هایی $w_{i,j}$ را بیابد به‌گونه‌ای که مربعِ فاصله‌ی بین $\mathbf{x}^{(i)}$ و $\sum_{j=1}^{m} \mathrm{w}_{\mathrm{I,j}}\, \mathrm{x}^{(\mathrm{j})}$ تا حد ممکن کم باشد، با این فرض که $w_{i,j} = 0$ اگر $\mathbf{x}^{(j)}$ یکی از k همسایه‌ی نزدیک $\mathbf{x}^{(i)}$ نباشد. بنابراین، گام اول الگوریتم LLE یک مسئله‌ی بهینه‌سازی با قُیود (constrained optimization)—مسئله‌ای که هدف آن پیدا کردن بهترین مقدار تابع هدف است، در حالی که محدودیت‌ها(Constraints) باید رعایت شوند—است که در معادله ۸-۴ آمده، جایی که $\mathbf{W}$ ماتریس وزن‌هاست و شامل تمام ضرایب $w_{i,j}$ می‌باشد. محدودیت دوم، به‌سادگی وزن‌های مربوط به هر نمونه‌ی آموزشی $\mathbf{x}^{(i)}$ را نرمال‌سازی می‌کند.

معادله ۸-۴. گام اول الگوریتم LLE: مدل‌سازی خطی روابط محلی

© 2025 Mohsen Zare. All rights reserved.

$$\widehat{W} = \text{argmin}_W \sum_{i=1}^{m} \left\| x^{(i)} - \sum_{j=1}^{m} w_{i,j}\, x^{(j)} \right\|^2$$

منوط به:

1. $w_{i,j} = 0$ if $x^{(j)}$ is not one of the knn of $x^{(i)}$
2. $\sum_{j=1}^{m} w_{i,j} = 1$ for all $i = 1,2,\cdots,m$

پس از این مرحله، ماتریس وزن $\widehat{\mathrm{W}}$ (که شامل وزن‌های $\widehat{w}_{i,j}$ است) روابط خطی محلی میان نمونه‌های آموزشی را در خود ثبت می‌کند. گام دوم این است که نمونه‌های آموزشی را به یک فضای d-بعدی نگاشت کنیم (که در آن $d < n$)، به‌گونه‌ای که این روابط محلی تا حد امکان حفظ شوند. اگر $\mathbf{z}^{(i)}$ تصویر $\mathbf{x}^{(i)}$ در این فضای d-بعدی باشد، آنگاه می‌خواهیم مربعِ فاصله‌ی بین $\mathbf{z}^{(i)}$ و $\sum_{j=1}^{m} \widehat{w}_{i.j}\, z^{(j)}$ تا حد ممکن کوچک باشد. این ایده منجر به مسئله‌ی بهینه‌سازی بدون قید (unconstrained optimization) می‌شود که در معادله‌ی ۸-۵ توصیف شده است. این مرحله بسیار شبیه به گام اول به نظر می‌رسد، اما با یک تفاوت کلیدی: به‌جای اینکه نمونه‌ها را ثابت نگه داریم و به‌دنبال وزن‌های بهینه باشیم، این‌بار وزن‌ها ثابت هستند و هدف، یافتن موقعیت بهینه‌ی تصویرشده‌ی نمونه‌ها در فضای کم‌بعد است. توجه داشته باشید که $\mathbf{Z}$ ماتریسی است که تمام بردارهای $\mathbf{z}^{(i)}$ (پروجکشن‌های نمونه‌ها در فضای کم‌بعد) را در خود جای داده است.

معادله ۸-۵.گام دوم الگوریتم LLE: کاهش ابعاد با حفظ روابط محلی

$$\hat{Z} = \text{argmin}_Z \sum_{i=1}^{m} \left\| z^{(i)} - \sum_{j=1}^{m} \widehat{w}_{i,j}\, z^{(j)} \right\|^2$$

پیاده‌سازی LLE در Scikit-Learn دارای پیچیدگی محاسباتی زیر است:

© 2025 Mohsen Zare. All rights reserved.

$O(m\ log(m)\ n\ log(k))$ برای یافتن k-همسایهٔ نزدیک، $O(mnk^3)$ برای بهینه‌سازی وزن‌ها، و $O(dm^2)$ برای ساخت نمایش کم‌بعد. متأسفانه، وجود m^2 در جمله آخر باعث می‌شود که این الگوریتم برای مجموعه داده‌های بسیار بزرگ به‌خوبی مقیاس‌پذیر نباشد.

همان‌طور که می‌بینید، LLE با تکنیک‌های پروجکشن تفاوت زیادی دارد و به‌مراتب پیچیده‌تر است، اما می‌تواند نمایش‌های بسیار بهتری در فضای با ابعاد پایین بسازد، به‌ویژه زمانی که داده‌ها غیرخطی باشند.

© 2025 Mohsen Zare. All rights reserved.

سایر تکنیک‌های کاهش ابعاد

پیش از آنکه این فصل را به پایان ببریم، بیایید نگاهی سریع به چند تکنیک محبوب دیگر کاهش ابعاد که در کتابخانه Scikit-Learn در دسترس هستند بیندازیم:

sklearn.manifold.MDS

«مقیاس‌بندی چندبعدی (*Multidimensional Scaling* یا MDS)» با هدف حفظ فواصل بین نمونه‌ها، ابعاد داده را کاهش می‌دهد. پروجِکشن تصادفی برای داده‌های با ابعاد بالا به خوبی عمل می‌کند، اما در داده‌های با ابعاد پایین عملکرد مناسبی ندارد.

sklearn.manifold.Isomap

«*Isomap*» یک گراف ایجاد می‌کند که در آن هر نمونه به نزدیک‌ترین همسایه‌هایش متصل می‌شود، سپس با حفظ فاصله‌های «ژئودزیک(*geodesic distances*)» (مسیرهای خمیده روی مَنیفولد)، ابعاد داده را کاهش می‌دهد. فاصله‌ی ژئودزیک بین دو گره در یک گراف، تعداد گره‌هایی است که در کوتاه‌ترین مسیر بین این دو گره قرار دارند.

sklearn.manifold.TSNE

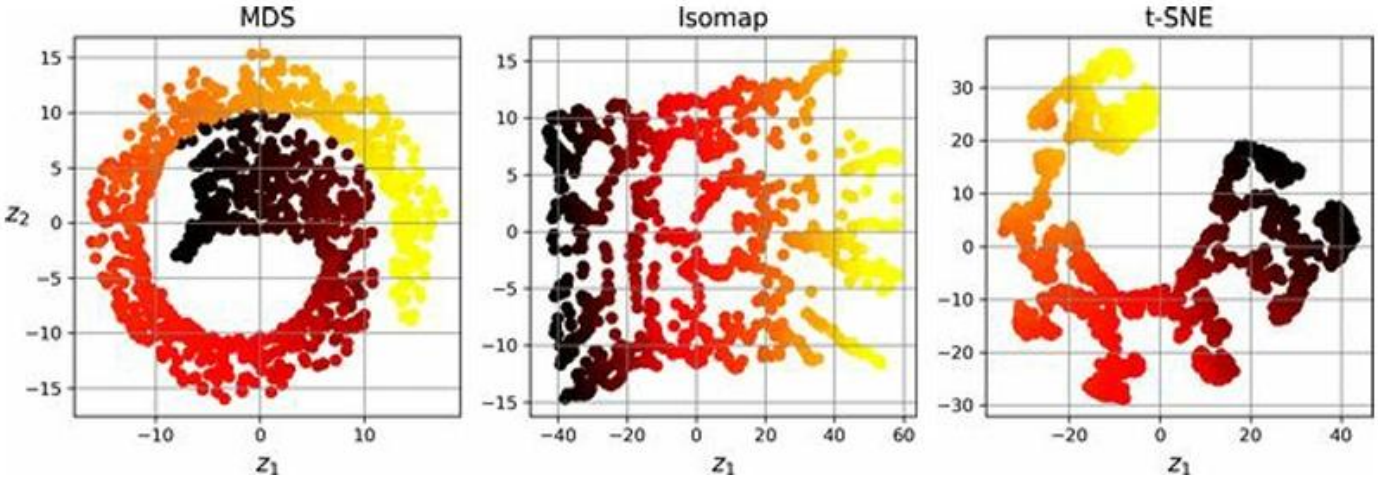
«تکنیک t-SNE به معنی توزیع تصادفی t برای جاسازی همسایگی (*t-distributed stochastic neighbor embedding*)» با کاهش بُعد داده‌ها تلاش می‌کند نمونه‌های مشابه را در کنار هم و نمونه‌های غیرمشابه را از هم دور نگه دارد. این روش عمدتاً برای مصورسازی(تجسم) استفاده می‌شود، به‌ویژه برای نمایش خوشه‌های نمونه‌ها در فضای با بُعد بالا. برای مثال، در تمرین‌های پایان این فصل از t-SNE استفاده خواهید کرد تا نقشه‌ای دوبعدی از تصاویر MNIST را بصری‌سازی کنید.

© 2025 Mohsen Zare. All rights reserved.

sklearn.discriminant_analysis.LinearDiscriminantAnalysis

«تحلیلِ تفکیک خطی (*Linear discriminant analysis* یا LDA)» یک الگوریتم طبقه‌بندی خطی است که در مرحله آموزش، محورهایی را می‌آموزد که بیشترین جداسازی را بین کلاس‌ها ایجاد می‌کنند. سپس می‌توان از این محورهای تفکیک‌کننده برای تعریف یک ابرصفحه استفاده کرد تا داده‌ها روی آن پروجکت شوند. مزیت این روش این است که این نوع پروجکشن، کلاس‌ها را تا حد ممکن از هم دور نگه می‌دارد. بنابراین LDA یک تکنیک مناسب برای کاهش ابعاد پیش از اجرای یک الگوریتم طبقه‌بندی دیگر است (مگر اینکه خودِ LDA به‌تنهایی کافی باشد).

شکل ۸-۱۱ نتایج MDS، Isomap و t-SNE را روی داده‌ی رولت سوئیسی نشان می‌دهد. MDS موفق می‌شود رولت سوئیسی را بدون از دست دادن انحنای کلی آن صاف کند، در حالی که Isomap این انحنا را کاملاً از بین می‌برد. بسته به وظیفه‌ی نهایی (downstream task)، حفظ ساختار در مقیاس بزرگ(large-scale structure) می‌تواند مفید یا مضر باشد. t-SNE عملکرد نسبتاً خوبی در تخت‌کردن رولت سوئیسی دارد، مقداری از انحنای آن را حفظ می‌کند و همچنین خوشه‌ها را برجسته می‌سازد، به‌طوری که رول را از هم جدا می‌کند. باز هم، این موضوع می‌تواند بسته به وظیفه‌ی نهایی خوب یا بد باشد.



شکل ۸-۱۱. استفاده از تکنیک‌های مختلف برای کاهش ابعاد رول سوئیسی به فضای دوبعدی

© 2025 Mohsen Zare. All rights reserved.

تمرین‌ها

۱. انگیزه‌های اصلی برای کاهش ابعاد یک مجموعه‌داده چیست؟ معایب اصلی آن کدام‌اند؟

۲. نفرین ابعاد بالا چیست؟

۳. آیا پس از کاهش ابعاد یک مجموعه‌داده، می‌توان این عملیات را معکوس کرد؟ اگر بله، چگونه؟ اگر نه، چرا؟

۴. آیا می‌توان از PCA برای کاهش ابعاد یک مجموعه‌داده‌ی شدیداً غیرخطی استفاده کرد؟

۵. فرض کنید روی یک مجموعه‌داده با ۱,۰۰۰ بُعد، تحلیل مؤلفه‌های اصلی (PCA) انجام می‌دهید و نسبت واریانس توضیح‌داده‌شده را روی ۹۵% تنظیم می‌کنید. مجموعه داده حاصل چند بُعد خواهد داشت؟

۶. در چه مواردی از PCA معمولی، PCA افزایشی، PCA تصادفی یا تصویرسازی تصادفی استفاده می‌کنید؟

۷. چطور می‌توانید عملکرد یک الگوریتم کاهش ابعاد را بر روی داده‌های خود ارزیابی کنید؟

۸. آیا منطقی است که دو الگوریتم کاهش ابعاد مختلف را به هم متصل کنید؟

۹. مجموعه داده MNIST را بارگذاری کنید (که در فصل ۳ معرفی شد) و آن را به مجموعه داده آموزشی و آزمون تقسیم کنید (اولین ۶۰,۰۰۰ نمونه برای آموزش، و ۱۰,۰۰۰ نمونه باقی‌مانده برای آزمون). یک مدلِ طبقه‌بندِ جنگل تصادفی روی مجموعه داده آموزش بدهید و زمان لازم برای آموزش را اندازه‌گیری کنید، سپس مدل به‌دست‌آمده را روی مجموعه دادهٔ آزمون ارزیابی کنید. سپس از روش PCA برای کاهش بُعد مجموعه داده استفاده کنید، به‌طوری که نسبت واریانس تبیین‌شده(explained variance) برابر با ۹۵% باشد. یک مدل جدید از طبقه‌بند جنگل تصادفی را روی مجموعه داده کاهش‌یافته آموزش دهید و ببینید چقدر زمان می‌برد. آیا زمان

© 2025 Mohsen Zare. All rights reserved.

آموزش خیلی سریع‌تر شده بود؟ سپس، طبقه‌بند را روی مجموعه آزمون ارزیابی کنید. چطور با طبقه‌بند قبلی مقایسه می‌شود؟ دوباره با استفاده از یک SGDClassifier امتحان کنید. الان چقدر PCA کمک کرده است؟

۱۰. از t-SNE برای کاهش ابعاد ۵,۰۰۰ تصویر اول از مجموعه داده‌ی MNIST به ۲ بعد استفاده کنید و نتیجه را با استفاده از Matplotlib ترسیم کنید. می‌توانید از یک نمودار پراکندگی (scatterplot) با ۱۰ رنگ مختلف برای نمایش هر کلاس هدف تصویر استفاده کنید. در غیر این صورت، می‌توانید هر نقطه در نمودار پراکندگی را با کلاس مربوطه هر نمونه (یک عدد از ۰ تا ۹) جایگزین کنید، یا حتی نسخه‌های کوچکتر از تصاویر ارقام خود را رسم کنید (اگر تمام ارقام را رسم کنید، نمودار خیلی شلوغ خواهد شد، بنابراین باید یا یک نمونه تصادفی رسم کنید یا تنها در صورتی یک نمونه را رسم کنید که هیچ نمونه دیگری در فاصله نزدیک به آن قبلاً رسم نشده باشد). باید یک بصری‌سازی(تجسم) زیبا با خوشه‌های جداشده از ارقام دریافت کنید. تلاش کنید از الگوریتم‌های کاهش ابعاد دیگری مانند PCA، LLE یا MDS استفاده کنید و نتایج و بصری‌سازی‌های حاصل را مقایسه کنید.

پاسخ‌های مربوط به این تمرین‌ها را می‌توانید در انتهای نوت‌بوک همین فصل، از طریق نشانی *https://homl.info/colab3* مشاهده کنید.

© 2025 Mohsen Zare. All rights reserved.

فصل ۹. تکنیک‌های یادگیری بدون نظارت

اگرچه بیشتر کاربردهای یادگیری ماشین امروز بر اساس یادگیری نظارت‌شده هستند (و به همین دلیل بیشتر سرمایه‌گذاری‌ها در این حوزه انجام می‌شود)، اما بیشتر داده‌های موجود بدون برچسب هستند: ما ویژگی‌های ورودی $\mathbf{X}$ را داریم، اما برچسب‌ها $\mathbf{y}$ را نداریم. یان لِکان، دانشمند کامپیوتر، جمله معروفی دارد که می‌گوید: «اگر هوش یک کیک باشد، یادگیری بدون نظارت خود کیک است، یادگیری نظارت‌شده مانند خامه روی کیک است و یادگیری تقویتی همان گیلاس روی کیک است.» به عبارت دیگر، یادگیری بدون نظارت پتانسیل عظیمی دارد که ما فقط کمی شروع به کشف آن کرده‌ایم.

فرض کنید می‌خواهید سیستمی بسازید که چند تصویر از هر کالا در یک خط تولید بگیرد و تشخیص دهد کدام کالاها معیوب هستند. شما می‌توانید به‌راحتی سیستمی بسازید که به‌طور خودکار تصاویر را بگیرد، و این ممکن است هر روز هزاران تصویر به شما بدهد. سپس می‌توانید در عرض چند هفته یک مجموعه داده نسبتاً بزرگ بسازید. اما صبر کنید، هیچ برچسبی وجود ندارد! اگر بخواهید یک مدل طبقه‌بندیِ باینری معمولی بسازید که پیش‌بینی کند آیا یک شیء معیوب است یا نه، باید هر تصویر را به‌صورت دستی برچسب‌گذاری کنید و آن را به عنوان "معیوب" یا "عادی" مشخص کنید. این امر عموماً نیاز به این دارد که کارشناسان انسانی بنشینند و وقت بگذارند و به‌طور دستی تمام تصاویر را بررسی

© 2025 Mohsen Zare. All rights reserved.

کنند. این یک کار طولانی، پرهزینه و خسته‌کننده است، بنابراین معمولاً فقط روی یک زیرمجموعه کوچک از تصاویر موجود انجام می‌شود. در نتیجه، مجموعه داده‌های برچسب‌خورده نسبتاً کوچک و عملکرد طبقه‌بند بسیار نااُمیدکننده خواهد بود. علاوه بر این، هر بار که شرکت تغییری در محصولات خود ایجاد کند، تمام این فرآیند باید از ابتدا آغاز شود. آیا عالی نمی‌شد اگر الگوریتم می‌توانست از داده‌های بدون برچسب استفاده کند و دیگر نیازی به برچسب‌گذاری دستی هر تصویر توسط انسان‌ها نباشد؟ در اینجاست که یادگیری بدون نظارت وارد عمل می‌شود.

در فصل ۸، به رایج‌ترین مسئله‌ی یادگیری بدون نظارت، یعنی کاهش ابعاد پرداخته‌ایم. در این فصل به چند مسئله‌ی دیگر در یادگیری بدون نظارت خواهیم پرداخت:

خوشه‌بندی(*Clustering*)

هدف از این کار این است که نمونه‌های مشابه را در کنار هم گروه‌بندی کنیم و «خوشه‌ها (*clusters*)» را تشکیل دهیم. خوشه‌بندی ابزاری عالی برای تحلیل داده‌ها، تقسیم‌بندی مشتریان، سیستم‌های توصیه‌گر، موتورهای جستجو، تقسیم‌بندی تصاویر، یادگیری نیمه‌نظارتی، کاهش ابعاد و بسیاری از کاربردهای دیگر است.

تشخیص ناهنجاری(*Anomaly detection*) که به آن شناسایی داده های پَرت (*outlier detection*) یا تشخیص نقاط دورافتاده از الگو کلی داده‌ها نیز گفته می‌شود

هدف این است که الگوی داده‌های «عادی» (normal) را یاد بگیریم و سپس از آن برای شناسایی موارد غیرعادی(abnormal یا *anomalous*) استفاده کنیم. این نمونه‌ها به نام‌های «داده‌های غیرمعمول یا ناهنجاری‌ها (*anomalies*) و یا موارد خارج از محدوده (*outliers*) و یا داده های پَرت» شناخته می‌شوند، در حالی که نمونه‌های عادی به نام «نقاط عادی یا داده‌های درون‌محدوده (*inliers*)»—و داده‌هایی که با الگوی غالب و میانگین داده‌ها سازگار هستند—شناخته می‌شوند. تشخیص ناهنجاری‌ها در انواع مختلفی از کاربردها مفید است، مانند شناسایی تقلب، تشخیص محصولات معیوب در تولید، شناسایی روندهای جدید در سری‌های زمانی، یا حذف داده‌های پَرت از یک مجموعه‌داده قبل از آموزش مدل دیگری، که می‌تواند عملکرد مدل نهایی را به طور قابل توجهی بهبود بخشد.

© 2025 Mohsen Zare. All rights reserved.

تخمینِ چگالی(*Density estimation*)

این مسئله‌ی تخمین «تابع چگالی احتمال (*probability density function* یا PDF)» فرآیندی تصادفی است که مجموعه‌داده‌ها را تولید کرده است. تخمین چگالی معمولاً برای شناسایی ناهنجاری‌ها استفاده می‌شود: نمونه‌هایی که در نواحی با چگالی بسیار پایین قرار دارند، احتمالاً ناهنجاری هستند. این روش برای تحلیل داده‌ها و مصورسازی (Visualization) نیز مفید است.

آماده برای یک کیک هستید؟ با دو الگوریتم خوشه‌بندی، k-means و DBSCAN شروع می‌کنیم، سپس مدل‌های مخلوطِ گاوسی—مدلی که داده‌ها را به‌صورت ترکیب چند توزیع گاوسی مستقل توصیف می‌کند—را بررسی خواهیم کرد و خواهیم دید که چگونه می‌توانند برای تخمین چگالی، خوشه‌بندی و شناسایی ناهنجاری‌ها استفاده شوند.

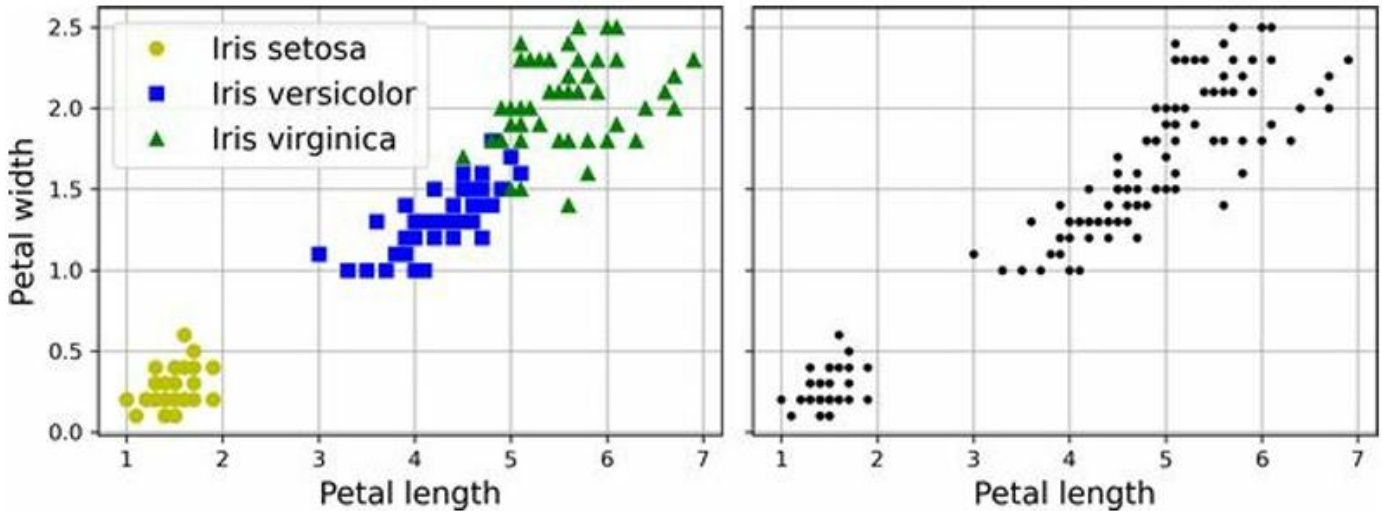
© 2025 Mohsen Zare. All rights reserved.

الگوریتم‌های خوشه‌بندی: k-means و DBSCAN

وقتی در حال پیاده‌روی در کوه‌ها هستید، به گیاهی برمی‌خورید که قبلاً آن را ندیده‌اید. به اطراف نگاه می‌کنید و چند گیاه دیگر مشاهده می‌کنید. آن‌ها دقیقاً مشابه نیستند، اما به اندازه‌ای شبیه به هم هستند که می‌توانید تشخیص دهید احتمالاً متعلق به همان گونه (یا حداقل همان جنس) هستند. شاید برای تشخیص اینکه آن گیاه متعلق به کدام گونه است نیاز به یک گیاه‌شناس داشته باشید، اما قطعاً برای شناسایی گروه‌هایی از اشیای مشابه، به یک کارشناس نیاز ندارید. این فرایند را «خوشه‌بندی (*Clustering*)» می‌نامند: مسئله شناسایی نمونه‌های مشابه و اختصاص آن‌ها به «خوشه‌ها(*clusters*)» یا گروه‌هایی از نمونه‌های مشابه است.

دقیقا مانند طبقه‌بندی، هر نمونه به یک گروه اختصاص داده می‌شود. اما برخلاف طبقه‌بندی، خوشه‌بندی یک مسئله‌ی بدون نظارت است. تصویر ۹-۱ را در نظر بگیرید: در سمت چپ، مجموعه داده زنبق (که در فصل ۴ معرفی شد) قرار دارد، که در آن هر نمونه با نشانگری متفاوت بر اساس گونه (species) خود (یعنی کلاس آن نمونه) نمایش داده شده است. این یک مجموعه داده برچسب‌خورده است که برای الگوریتم‌های طبقه‌بندی مانند رگرسیون لجستیک(LRCs)، ماشین‌های بردار پشتیبان (SVMs) یا طبقه‌بندهای جنگل تصادفی(RFCs) مناسب است. در سمت راست همان مجموعه داده است، اما بدون برچسب‌ها، بنابراین دیگر نمی‌توان از الگوریتم‌های طبقه‌بندی استفاده کرد. این‌جاست که الگوریتم‌های خوشه‌بندی وارد عمل می‌شوند: بسیاری از آن‌ها به‌راحتی می‌توانند خوشه پایین سمتِ چپ را شناسایی کنند. این با چشم خودمان هم به راحتی قابل مشاهده است، اما چندان واضح نیست که خوشه بالای سمت راست از دو زیرخوشه متمایز و مجزا تشکیل شده باشد. با این حال، مجموعه داده دو ویژگی اضافی (طول و عرض کاسبرگ) دارد که در اینجا نمایش داده نشده‌اند، و الگوریتم‌های خوشه‌بندی می‌توانند به خوبی از همه ویژگی‌ها استفاده کنند، بنابراین در واقع این الگوریتم‌ها سه خوشه را به‌خوبی شناسایی می‌کنند (مثلاً با استفاده از مدل مخلوط گاوسی، فقط ۵ نمونه از ۱۵۰ نمونه به خوشه اشتباهی تعلق دارند).

© 2025 Mohsen Zare. All rights reserved.



شکل ۹-۱. طبقه‌بندی (چپ) در مقابل خوشه‌بندی (راست)

خوشه‌بندی در کاربردهای متنوعی استفاده می‌شود، از جمله:

تقسیم‌بندی مشتریان(*Customer segmentation*)

شما می‌توانید مشتریان خود را بر اساس خریدها و فعالیت‌هایشان در وب‌سایتتان دسته بندی(خوشه‌بندی) کنید. این کار برای درک بهتر نیازها و ویژگی‌های مشتریان مفید است، تا بتوانید محصولات و کمپین‌های بازاریابی خود را متناسب با هر بخش از مشتریان تنظیم کنید. به عنوان مثال، تقسیم‌بندی مشتریان می‌تواند در «سیستم‌های توصیه‌گر(*recommender systems*)» مفید باشد تا محتوایی را که کاربران دیگر در همان خوشه پسندیده‌اند، پیشنهاد دهد.

تحلیل داده‌ها(*Data analysis*)

وقتی که یک مجموعه داده جدید را تجزیه و تحلیل می‌کنید، ممکن است مفید باشد که ابتدا یک الگوریتم خوشه‌بندی اجرا کرده و سپس هر خوشه را به‌طور جداگانه تحلیل کنید.

کاهش ابعاد(*Dimensionality reduction*)

© 2025 Mohsen Zare. All rights reserved.

پس از این که یک مجموعه داده خوشه‌بندی شد، معمولاً می‌توان میزان وابستگیِ هر نمونه به هر خوشه را اندازه‌گیری کرد؛ «نزدیکی یا وابستگی (*affinity*)» به هر معیاری اطلاق می‌شود که نشان دهد یک نمونه چقدر به یک خوشه تعلق دارد. سپس بردار ویژگی هر نمونه ($\mathbf{x}$) می‌تواند با بردار وابستگی‌های آن به خوشه‌ها جایگزین شود. اگر k خوشه وجود داشته باشد، آنگاه این بردار k-بُعدی خواهد بود. این بردار جدید معمولاً ابعاد بسیار کمتری نسبت به بردار ویژگی اصلی دارد، اما می‌تواند اطلاعات کافی برای پردازش‌های بعدی را حفظ کند.

مهندسی ویژگی(*Feature engineering*)

اغلب، وابستگی‌های خوشه‌ای می‌توانند به عنوان ویژگی‌های اضافی مفید باشند. به عنوان مثال، در فصل ۲ از الگوریتم k-means برای اضافه کردن ویژگی‌های وابستگی خوشه‌ای جغرافیایی به مجموعه داده مسکن کالیفرنیا استفاده کردیم، و این ویژگی‌ها به بهبود عملکرد ما کمک کردند.

تشخیص ناهنجاری (*Anomaly detection*) که به آن شناساییِ نقاط پَرت (*outlier detection*) نیز گفته می‌شود

هر نمونه‌ای که ارتباط ضعیفی(میل ترکیبی کمی) با همه‌ی خوشه‌ها داشته باشد، احتمالاً یک ناهنجاری(داده‌ی غیرمعمول یا پَرت) است. برای مثال، اگر کاربران وب‌سایت خود را بر اساس رفتارشان خوشه‌بندی کرده‌اید، می‌توانید کاربرانی را که رفتار غیرمعمولی دارند شناسایی کنید، مانند تعداد غیرعادی درخواست‌ها در هر ثانیه.

یادگیری نیمه نظارتی(*Semi-supervised learning*)

اگر فقط چند برچسب (label) دارید، می‌توانید خوشه‌بندی انجام دهید و برچسب‌ها را به تمام نمونه‌های موجود در همان خوشه منتقل کنید. این تکنیک می‌تواند تعداد برچسب‌های موجود برای یک الگوریتم یادگیری نظارت‌شده بعدی را به‌طور قابل‌توجهی افزایش دهد و در نتیجه عملکرد آن را بهبود بخشد.

© 2025 Mohsen Zare. All rights reserved.

موتورهای جستجو(*Search engines*)

برخی از موتورهای جستجو به شما این امکان را می‌دهند که تصاویری مشابه با یک تصویر مرجع جستجو کنید. برای ساخت چنین سیستمی، ابتدا باید یک الگوریتم خوشه‌بندی را روی تمام تصاویر موجود در پایگاه داده خود اعمال کنید؛ تصاویر مشابه در یک خوشه قرار می‌گیرند. سپس زمانی که یک کاربر تصویری مرجع را ارائه می‌دهد، تمام چیزی که نیاز دارید این است که از مدل خوشه‌بندی آموزش‌دیده استفاده کنید تا خوشه‌ی مربوط به این تصویر را پیدا کنید، و سپس به سادگی می‌توانید تمام تصاویر موجود در آن خوشه را بازگردانید.

بخش‌بندی تصویر(*Image segmentation*)

با خوشه‌بندی پیکسل‌ها بر اساس رنگ آن‌ها و سپس جایگزینی رنگ هر پیکسل با رنگ میانگین خوشه‌اش، می‌توان تعداد رنگ‌های مختلف در یک تصویر را به طور قابل‌توجهی کاهش داد. بخش‌بندی یا قطعه‌بندی تصویر در بسیاری از سیستم‌های شناسایی و ردیابی اشیاء استفاده می‌شود، زیرا این کار شناسایی محدوده یا کانتور(contour)—برای مشخص کردن محدوده یک شیء یا منطقه خاص در تصویر—را راحت‌تر می‌کند.

تعریف واحدی برای خوشه وجود ندارد: این موضوع به بافت و زمینه بستگی دارد، و الگوریتم‌های مختلف خوشه‌های مختلفی را شناسایی می‌کنند. برخی از الگوریتم‌ها به دنبال نمونه‌هایی می‌گردند که حول یک نقطه خاص، به نام «نقطه مرکز هندسی یا مرکز خوشه (*centroid*)»، متمرکز شده‌اند. برخی دیگر به دنبال نواحی پیوسته از نمونه‌های متراکم می‌گردند: این خوشه‌ها می‌توانند هر شکلی به خود بگیرند. برخی الگوریتم‌ها سلسله‌مراتبی هستند و به دنبال خوشه‌هایی از خوشه‌ها می‌گردند. و فهرست همینطور ادامه دارد.

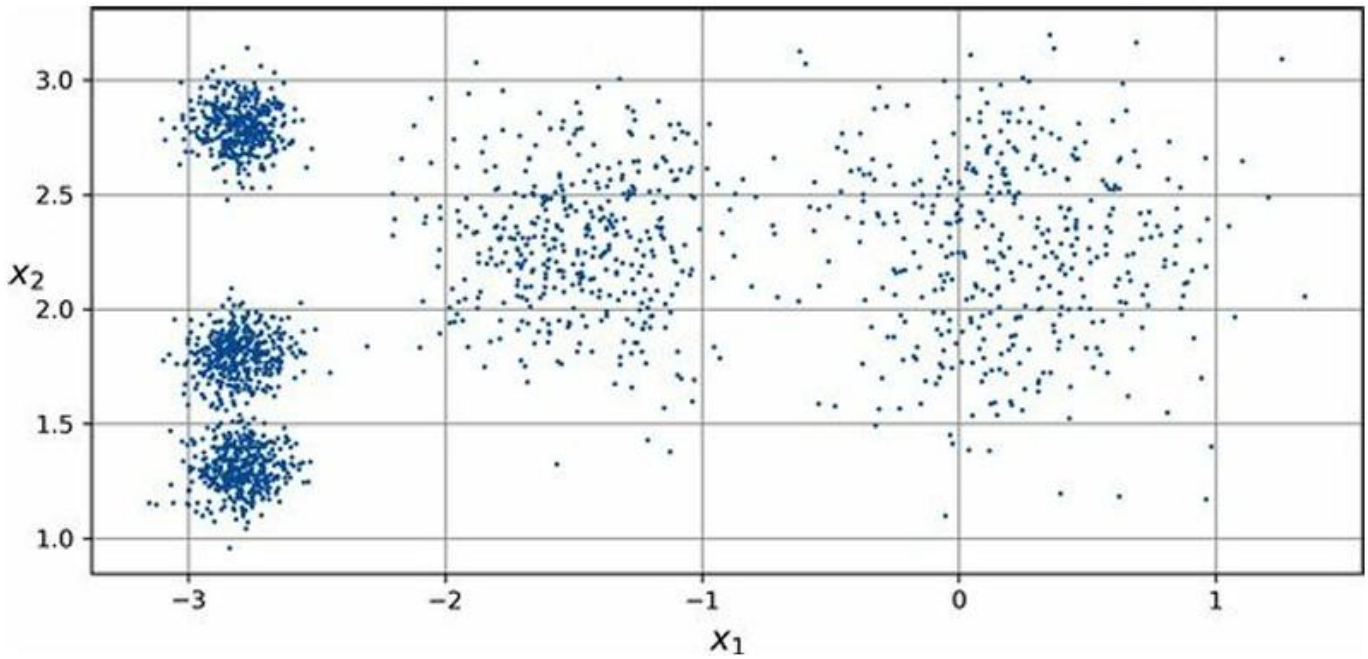
© 2025 Mohsen Zare. All rights reserved.

در این بخش، به بررسی دو الگوریتم محبوب خوشه‌بندی، یعنی k-means و DBSCAN خواهیم پرداخت و برخی از کاربردهای آن‌ها مانند کاهش ابعاد غیرخطی، یادگیری نیمه‌نظارت‌شده و شناسایی ناهنجاری‌ها را بررسی خواهیم کرد.

© 2025 Mohsen Zare. All rights reserved.

k-means

تصور کنید یک مجموعه داده بدون برچسب مانند آنچه در شکل ۹-۲ نشان داده شده است، دارید: شما به‌وضوح پنج خوشه یا حُباب(blob) از نمونه‌ها را مشاهده می‌کنید. الگوریتم k-means یک الگوریتم ساده است که قادر است چنین مجموعه داده‌ای را به‌سرعت و به‌طور کارآمد خوشه‌بندی کند، معمولاً تنها در چند دور تکرار. این الگوریتم توسط «اِستوارت لُوید» در آزمایشگاه‌های بل در سال ۱۹۵۷ به‌عنوان یک تکنیک برای مدولاسیون پالس-کد (pulse-code modulation) پیشنهاد شد، اما فقط در سال ۱۹۸۲ و آن‌هم به‌طور عمومی منتشر گردید. در سال ۱۹۶۵، «اِدوارد دَبلیو. فورگی» تقریباً همان الگوریتم را منتشر کرد، بنابراین الگوریتم k-means گاهی اوقات به‌عنوان الگوریتم لوید-فورگی نیز شناخته می‌شود.



شکل ۹-۲. یک مجموعه داده بدون برچسب که از پنج خوشه از نمونه‌ها تشکیل شده است

بیایید یک مدل خوشه‌بندی k-means روی این مجموعه داده آموزش دهیم. این الگوریتم سعی خواهد کرد مرکز هر خوشه را پیدا کرده و هر نمونه را به نزدیک‌ترین خوشه اختصاص دهد:

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
```

© 2025 Mohsen Zare. All rights reserved.

```
X, y = make_blobs([...])  # make the blobs: y contains the cluster IDs, but we
            # will not use them; that's what we want to predict
k = 5
kmeans = KMeans(n_clusters=k, random_state=42)
y_pred = kmeans.fit_predict(X)
```

توجه داشته باشید که باید تعداد خوشه‌ها (k) را که الگوریتم باید پیدا کند، مشخص کنید. در این مثال، از روی داده‌ها به راحتی می‌توان فهمید که k باید برابر با ۵ تنظیم شود، اما به طور کلی این کار همیشه اینقدر ساده نیست. در ادامه در این مورد بیشتر صحبت خواهیم کرد. هر نمونه به یکی از پنج خوشه اختصاص خواهد یافت. در زمینه خوشه‌بندی، برچسبِ یک نمونه، اندیسِ خوشه‌ای است که الگوریتم، آن نمونه را به آن اختصاص می‌دهد؛ این نباید با برچسب‌های کلاس در دسته‌بندی اشتباه گرفته شود، که به‌عنوان هدف‌ها استفاده می‌شوند (به یاد داشته باشید که خوشه‌بندی یک مسئله‌ی یادگیری بدون نظارت است). نمونه‌ی KMeans برچسب‌های پیش‌بینی‌شده‌ی نمونه‌هایی که روی آن‌ها آموزش دیده است را حفظ می‌کند، که از طریق متغیر labels_instance در دسترس است.

```
>>> y_pred
array([4, 0, 1, ..., 2, 1, 0], dtype=int32)
>>> y_pred is kmeans.labels_
True
```

همچنین می‌توانیم نگاهی به پنج مرکز خوشه‌ای (centroid) بیندازیم که الگوریتم پیدا کرده است:

```
>>> kmeans.cluster_centers_
array([[-2.80389616,  1.80117999],
       [ 0.20876306,  2.25551336],
       [-2.79290307,  2.79641063],
       [-1.46679593,  2.28585348],
       [-2.80037642,  1.30082566]])
```

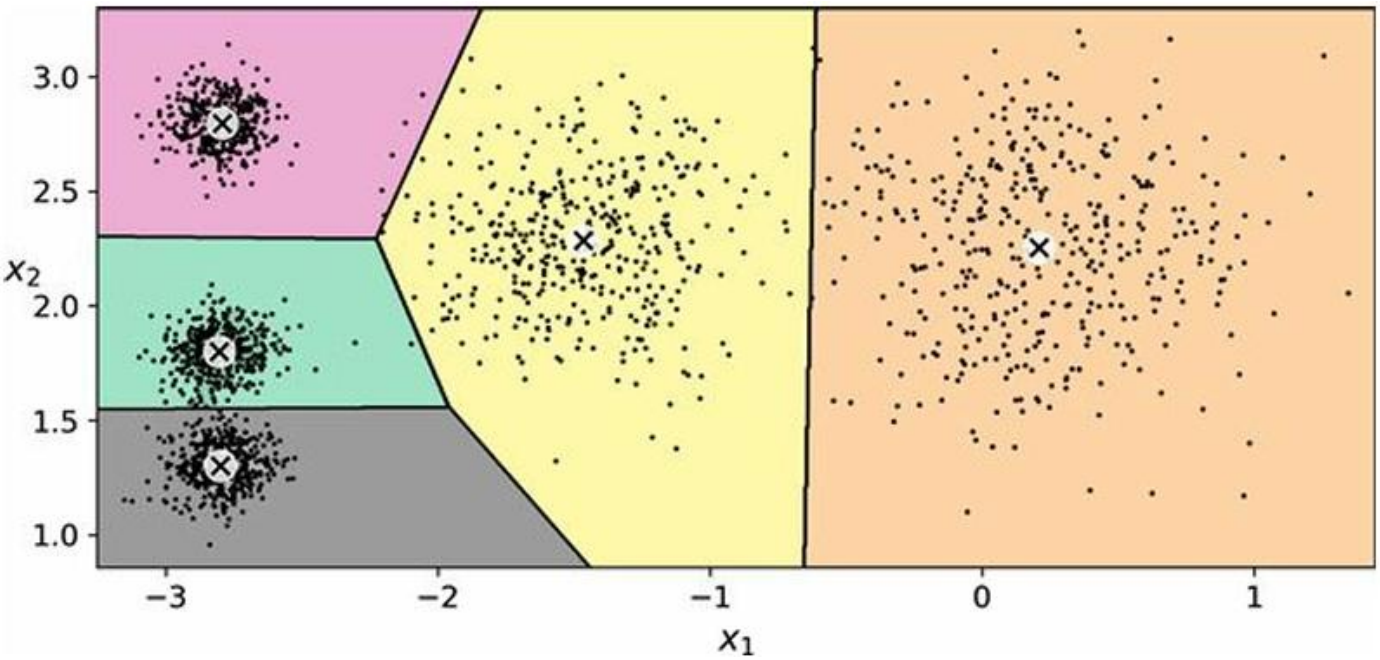
شما به‌راحتی می‌توانید نمونه‌های جدید را به خوشه‌ای که مرکز هندسی آن نزدیک‌تر است، اختصاص دهید:

```
>>> import numpy as np
```

© 2025 Mohsen Zare. All rights reserved.

```
>>> X_new = np.array([[0, 2], [3, 2], [-3, 3], [-3, 2.5]])
>>> kmeans.predict(X_new)
array([1, 1, 2, 2], dtype=int32)
```

اگر مرزهای تصمیم‌گیری خوشه‌ها را رسم کنید، یک تقسیم‌بندیِ وُرونوی(Voronoi tessellation) خواهید داشت: در شکل ۹-۳، هر مرکز خوشه با یک X نشان داده شده است.



شکل ۹-۳. مرزهای تصمیم‌گیری k-means (تقسیم‌بندی ورونوی)

بیشتر نمونه‌ها به وضوح به خوشه‌های مناسب تخصیص داده شدند، اما چند نمونه احتمالاً اشتباهاً برچسب‌گذاری شده‌اند، به‌ویژه نزدیک به مرز بین خوشه بالا سمت چپ و خوشه مرکزی. در واقع، الگوریتم k-means زمانی که خوشه‌ها قطرهای بسیار متفاوتی دارند، عملکرد خوبی ندارد، زیرا تنها چیزی که در هنگام اختصاص یک نمونه به یک خوشه برای آن مهم است، فاصله آن از مرکزِ هندسیِ خوشه است.

به جای اختصاص دادن هر نمونه به یک خوشه مشخص، که به آن «خوشه‌بندی سخت (*hard clustering*)» گفته می‌شود، ممکن است مفید باشد که به هر نمونه امتیازی(score) به ازای هر خوشه اختصاص داده شود، که به آن «خوشه‌بندی نرم (*soft clustering*)» گفته می‌شود. این امتیاز می‌تواند فاصله بین نمونه و مرکز خوشه یا یک امتیاز شباهت (یا وابستگی یا نزدیکی) باشد، مانند تابع پایه شعاعیِ گاوسی (Gaussian radial basis function) که در فصل ۲ استفاده کردیم. در

© 2025 Mohsen Zare. All rights reserved.

کلاس KMeans، متد ()transform فاصله هر نمونه را از هر یک از مراکز هندسیِ خوشه اندازه‌گیری می‌کند:

```
>>> kmeans.transform(X_new).round(2)
array([[2.81, 0.33, 2.9 , 1.49, 2.89],
       [5.81, 2.8 , 5.85, 4.48, 5.84],
       [1.21, 3.29, 0.29, 1.69, 1.71],
       [0.73, 3.22, 0.36, 1.55, 1.22]])
```

در این مثال، اولین نمونه در X_new فاصله‌ای معادل ۲٫۸۱ از اولین مرکز خوشه، ۰٫۳۳ از دومین مرکز خوشه، ۲٫۹۰ از سومین مرکز خوشه، ۱٫۴۹ از چهارمین مرکز خوشه و ۲٫۸۹ از پنجمین مرکز خوشه دارد.

اگر یک مجموعه داده با ابعاد بالا داشته باشید و آن را به این روش تبدیل کنید، در نهایت یک مجموعه داده با ابعادِ k خواهید داشت: این تبدیل می‌تواند یک تکنیک کاهش ابعاد غیرخطی بسیار کارآمد باشد. به‌طور جایگزین، می‌توانید از این فاصله‌ها به‌عنوان ویژگی‌های اضافی برای آموزش مدل دیگری استفاده کنید، همان‌طور که در فصل ۲ دیدیم.

الگوریتم k-means

پس، الگوریتم چطور کار می‌کند؟ خوب، فرض کنید که مراکز هندسیِ خوشه ها را به شما داده‌اند. شما به‌راحتی می‌توانید تمام نمونه‌های موجود در مجموعه داده را با انتساب هرکدام به خوشه‌ای که مرکز هندسیِ آن به نمونه نزدیک‌تر است، برچسب‌گذاری کنید. برعکس، اگر تمام برچسب‌های نمونه‌ها را داشته باشید، می‌توانید به‌راحتی مرکز هر خوشه را با محاسبه میانگین نمونه‌های آن خوشه پیدا کنید. اما نه برچسب‌ها و نه مراکز خوشه‌ها به شما داده شده است، پس چطور می‌توانید پیش بروید؟ شروع کنید به قرار دادن مراکز خوشه‌ها به صورت تصادفی (مثلاً با انتخاب k نمونه به طور تصادفی از مجموعه داده و استفاده از موقعیت‌های آن‌ها به عنوان مراکزِ خوشه‌ها). سپس نمونه‌ها را برچسب‌گذاری کنید، مرکزهای خوشه‌ها را به‌روزرسانی کنید، دوباره نمونه‌ها را برچسب‌گذاری کنید، مرکزهای خوشه‌ها را به‌روزرسانی کنید و این کار را ادامه دهید تا زمانی که مرکزهای خوشه‌ها دیگر جابه‌جا نشوند. الگوریتم تضمین شده است که در تعداد محدودی گام (معمولاً بسیار کم) همگرا

© 2025 Mohsen Zare. All rights reserved.

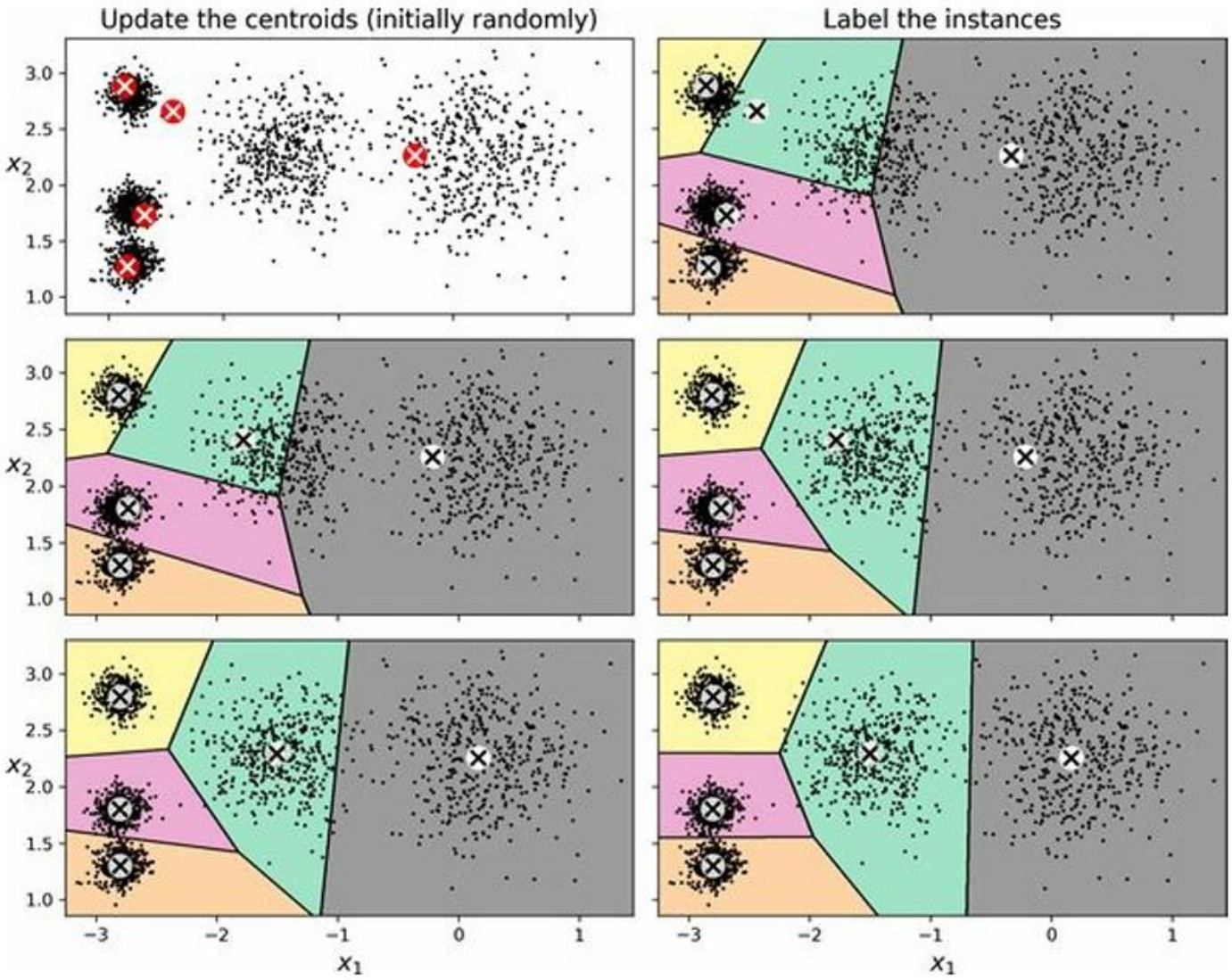
می‌شود. این به این دلیل است که مربعات میانگین فاصله‌ی(mean squared distance) بین نمونه‌ها و نزدیک‌ترین مرکزها در هر گام فقط می‌تواند کاهش یابد و از آنجا که این مقدار نمی‌تواند منفی شود، تضمین می‌شود که الگوریتم همگرا خواهد شد.

شما می‌توانید الگوریتم را در عمل در شکل ۹-۴ مشاهده کنید: مراکز به‌طور تصادفی مقداردهی اولیه می‌شوند (بالا سمت چپ)، سپس نمونه‌ها برچسب‌گذاری می‌شوند (بالا سمت راست)، سپس مراکز هندسی خوشه‌ها به‌روز می‌شوند (مرکز سمت چپ)، نمونه‌ها دوباره برچسب‌گذاری می‌شوند (مرکز سمت راست) و همینطور ادامه می‌یابد. همانطور که مشاهده می‌کنید، تنها در سه دور تکرار، الگوریتم به یک خوشه‌بندی که به‌نظر نزدیک به بهینه است، دست یافته است.

نکته

پیچیدگی محاسباتی الگوریتم معمولاً به‌طور خطی با تعداد نمونه‌ها (m)، تعداد خوشه‌ها (k)، و تعداد ابعاد (n) ارتباط دارد. با این حال، این موضوع فقط زمانی صحت دارد که داده‌ها ساختار خوشه‌بندی داشته باشند. اگر داده‌ها چنین ساختاری نداشته باشند، در بدترین حالت پیچیدگی ممکن است به‌طور نمایی با تعداد نمونه‌ها افزایش یابد. در عمل، این موضوع به‌ندرت اتفاق می‌افتد و الگوریتم k-means معمولاً یکی از سریع‌ترین الگوریتم‌های خوشه‌بندی است.

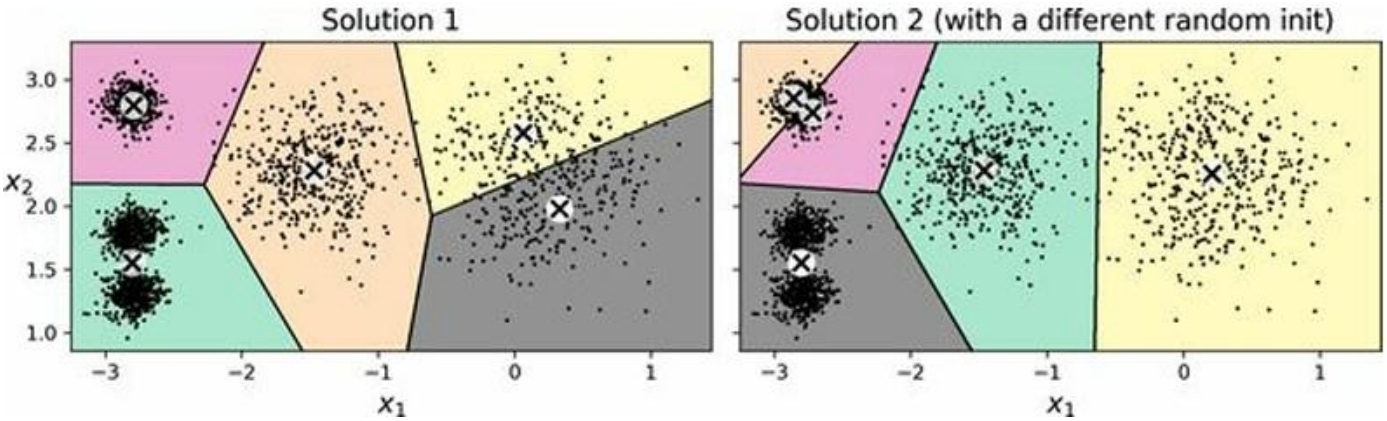
© 2025 Mohsen Zare. All rights reserved.



شکل ۹-۴. الگوریتم k-means

اگرچه الگوریتم تضمین شده است که همگرا می‌شود، اما ممکن است به راه‌حل یا جواب درست همگرا نشود (یعنی ممکن است به یک بهینه‌ی محلی(local optimum) همگرا شود): این که آیا الگوریتم به راه‌حل صحیح همگرا می‌شود یا نه، بستگی به نحوه‌ی مقداردهی اولیه‌ی مراکز خوشه‌ها دارد. شکل ۹-۵ دو راه‌حل غیر بهینه(suboptimal) را نشان می‌دهد که الگوریتم ممکن است به آن‌ها همگرا شود اگر در مرحله‌ی مقداردهی اولیه‌ی تصادفی شانس نیاورید.

© 2025 Mohsen Zare. All rights reserved.



شکل ۹-۵. راه‌حل‌های غیر بهینه به دلیل مقداردهی اولیه تصادفی نامناسب مراکز خوشه ها

بیایید به چند روش نگاه کنیم که می‌توانید این ریسک را با بهبود مقداردهی اولیه مرکزها کاهش دهید.

روش‌های مقداردهی اولیه مرکز هندسی خوشه

اگر به طور تقریبی بدانید که مرکزها باید کجا قرار گیرند (برای مثال، اگر قبلاً از یک الگوریتم خوشه‌بندی دیگر استفاده کرده‌اید)، می‌توانید هایپرپارامتر init را روی یک آرایه NumPy که شامل لیست مرکزها باشد، تنظیم کنید و مقدار n_init را برابر ۱ قرار دهید:

```
good_init = np.array([[-3, 3], [-3, 2], [-3, 1], [-1, 2], [0, 2]])
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1, random_state=42)
kmeans.fit(X)
```

راه‌حل دیگر این است که الگوریتم را چندین بار با تنظیمات اولیه تصادفی مختلف اجرا کنید و بهترین راه‌حل را نگه دارید. تعداد تنظیمات اولیه تصادفی توسط پارامتر n_init کنترل می‌شود: به‌طور پیش‌فرض این مقدار برابر با ۱۰ است، به این معنی که کل الگوریتمی که قبلاً توضیح داده شد، ۱۰ بار هنگام فراخوانی fit() اجرا می‌شود و Scikit-Learn بهترین راه‌حل را نگه می‌دارد. اما دقیقاً چطور می‌فهمد که کدام راه‌حل بهترین است؟ از یک معیار عملکرد (performance metric) استفاده می‌کند! این معیار به نام «اینرسیِ مدل (*model's inertia*)» شناخته می‌شود، که مجموع مربعات فواصل بین نمونه‌ها و نزدیک‌ترین مراکز خوشه‌ی آن‌هاست. این مقدار تقریباً برابر است با ۲۱۹/۴ برای مدل سمت چپ در شکل ۹-۵، ۲۵۸/۶ برای مدل سمت راست در همان شکل، و تنها ۲۱۱/۶ برای مدل موجود در شکل ۹-۳. کلاس KMeans این الگوریتم را به تعداد n_init بار اجرا می‌کند و مدلی را نگه

© 2025 Mohsen Zare. All rights reserved.

می‌دارد که کمترین میزان اینرسی را داشته باشد. در این مثال، مدلی که در شکل ۹-۳ نمایش داده شده انتخاب خواهد شد (مگر اینکه به‌طور بسیار نادری در تمام n_init مقداردهی‌های تصادفی متوالی بدشانس باشیم). اگر کنجکاو باشید، مقدار اینرسیِ یک مدل از طریقِ متغیر نمونه‌ای (instance variable) به نام inertia_ قابل دسترسی است:

```
>>> kmeans.inertia_
211.59853725816836
```

متد score() مقدار منفی اینرسی را برمی‌گرداند (منفی است چون متد score() در Scikit-Learn باید همواره از قانون «هرچه بزرگ‌تر، بهتر» پیروی کند: اگر یک پیش‌بینی‌کننده بهتر از دیگری باشد، متدِ score() مربوط به آن، باید مقدار امتیازِ بیشتری را بازگرداند):

```
>>> kmeans.score(X)
-211.5985372581684
```

یک بهبود مهم برای الگوریتم k-means، به نام *k-means*++، در مقاله‌ای در سال ۲۰۰۶ توسط «دِیوید آرتور» و «سِرگِئی واسیلویتسکی» پیشنهاد شد. آن‌ها یک گام اولیه‌ی هوشمندانه‌تر برای مقداردهی اولیه معرفی کردند که معمولاً مراکز خوشه‌ای را انتخاب می‌کند که از یکدیگر فاصله‌ی زیادی دارند. این بهبود باعث می‌شود الگوریتم k-means با احتمال بسیار کمتری به یک راه‌حل نامطلوب و غیر بهینه (suboptimal solution) همگرا شود. در این مقاله نشان داده شد که محاسبات اضافی لازم برای گام مقداردهی اولیه هوشمند ارزشش را دارد، زیرا این روش به طور قابل توجهی تعداد دفعات اجرای الگوریتم برای یافتن بهترین راه‌حل را کاهش می‌دهد. الگوریتم مقداردهی اولیه *k-means*++ به این صورت عمل می‌کند:

۱. یک مرکز خوشه $\mathbf{c}^{(i)}$ را انتخاب کن که به‌صورت یکنواخت و تصادفی از میان مجموعه داده انتخاب شده باشد.

۲. یک مرکز خوشه جدید $\mathbf{c}^{(i)}$ را انتخاب کن، به این صورت که یک نمونه $\mathbf{x}^{(\mathbf{i})}$ را با احتمالی برابر با $D\left(x^{(i)}\right)^2 / \sum_{j=1}^{m} D\left(x^{(j)}\right)^2$ انتخاب کنی؛ در اینجا $D\left(x^{(i)}\right)$ فاصله نمونه $\mathbf{x}^{(i)}$ تا نزدیک‌ترین مرکز خوشه‌ای است که پیش‌تر انتخاب شده است. این توزیع احتمالاتی باعث می‌شود

© 2025 Mohsen Zare. All rights reserved.

نمونه‌هایی که فاصله بیشتری از مراکز خوشه‌ای انتخاب‌شده دارند، با احتمال بیشتری به‌عنوان مرکز جدید انتخاب شوند.

۳. مرحله‌ی قبلی را تکرار کنید تا زمانی که همه‌ی k مرکزِ هندسی انتخاب شوند.

کلاس KMeans به‌طور پیش‌فرض از این روش مقداردهی اولیه استفاده می‌کند.

نسخه‌ی k-means شتاب‌یافته و نسخه‌ی k-means مینی دسته ای

بهبود دیگری برای الگوریتم k-means در مقاله‌ای از چارلز اِلکان در سال ۲۰۰۳ پیشنهاد شد. در برخی مجموعه داده‌های بزرگ با خوشه‌های زیاد، این الگوریتم را می‌توان با اجتناب از محاسبات غیرضروری فاصله، سریع‌تر اجرا کرد. اِلکان با استفاده از نابرابری مثلثی (یعنی اینکه همیشه خط مستقیم کوتاه‌ترین فاصله بین دو نقطه است) و با دنبال‌کردن کَران‌های بالا و پایین برای فاصله بین نمونه‌ها و مرکزهای خوشه، به این بهبود دست یافت. با این حال، الگوریتم اِلکان همیشه آموزش را سریع‌تر نمی‌کند و گاهی حتی ممکن است به‌طور قابل‌توجهی آموزش را کندتر کند؛ این موضوع به نوع داده‌ها بستگی دارد. با این وجود، اگر بخواهید آن را امتحان کنید، این تنظیم "algorithm="elkan را قرار دهید.

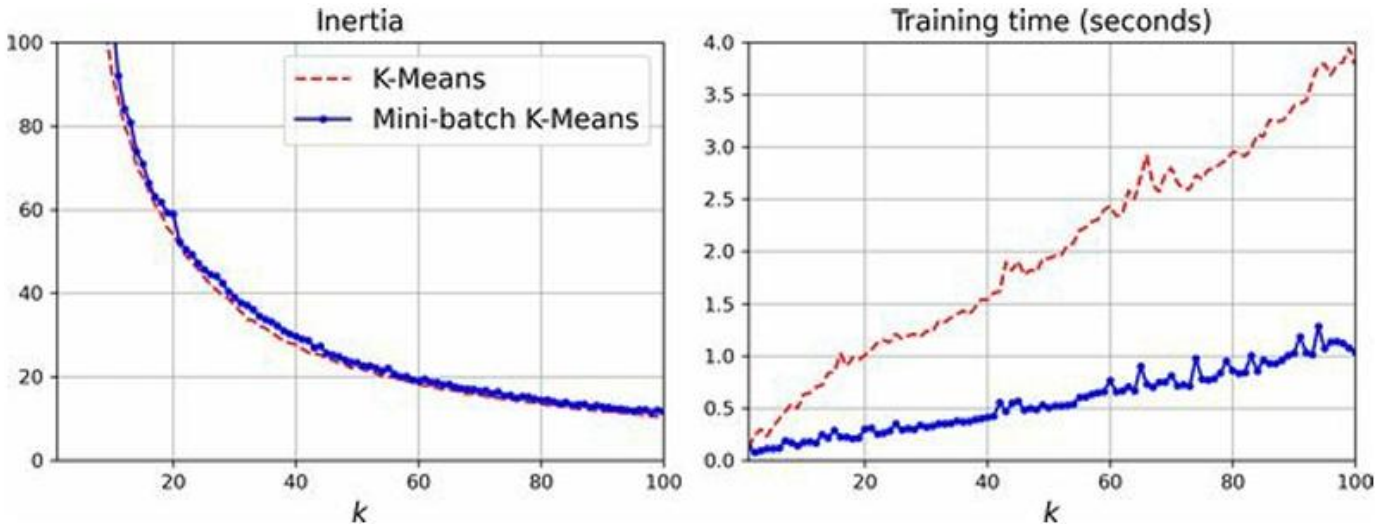
یک نسخه مهم دیگر از الگوریتم k-means در مقاله‌ای در سال ۲۰۱۰ توسط دیوید اسکالی ارائه شد. به جای استفاده از کل مجموعه‌داده در هر تکرار، این الگوریتم قادر است از مینی‌دسته ای‌ها(دسته های کوچک) استفاده کند و در هر مرحله، مراکز خوشه‌ها را به‌طور جزئی جابجا کند. این کار باعث افزایش سرعت الگوریتم می‌شود (معمولاً سه تا چهار برابر) و امکان خوشه‌بندی مجموعه‌داده‌های بسیار بزرگ که در حافظه جا نمی‌شوند را فراهم می‌کند. Scikit-Learn این الگوریتم را در کلاس MiniBatchKMeans پیاده‌سازی کرده است که می‌توانید مانند کلاس KMeans از آن استفاده کنید:

```
from sklearn.cluster import MiniBatchKMeans

minibatch_kmeans = MiniBatchKMeans(n_clusters=5, random_state=42)
minibatch_kmeans.fit(X)
```

© 2025 Mohsen Zare. All rights reserved.

اگر مجموعه‌داده در حافظه جا نشود، ساده‌ترین راه استفاده از کلاس memmap است، مانند کاری که در فصل ۸ برای incremental PCA انجام دادیم. به‌طور جایگزین، می‌توانید هر بار یک مینی‌دسته (mini-batch) را به متد partial_fit() ارسال کنید، اما این کار نیاز به تلاش بیشتری دارد چون باید چندین بار مقداردهی اولیه انجام داده و بهترین نتیجه را خودتان انتخاب کنید.

اگرچه الگوریتم mini-batch k-means بسیار سریع‌تر از الگوریتم معمولی k-means است، اما معمولاً اینرسی آن کمی ضعیف‌تر است. این موضوع را می‌توانید در شکل ۹-۶ ببینید: نمودار سمت چپ اینرسی‌های مدل‌های k-means مینی‌دسته‌ای و k-means معمولی را که روی همان داده‌های پنج خوشه ای قبلی با تعداد خوشه‌های مختلف k آموزش داده شده‌اند، مقایسه می‌کند. تفاوت بین دو منحنی، کوچک اما قابل مشاهده است. در نمودار سمت راست، می‌توانید ببینید که الگوریتم k-means مینی‌دسته‌ای تقریباً ۳٫۵ برابر سریع‌تر از الگوریتم k-means معمولی روی این مجموعه داده عمل می‌کند.



شکل ۹-۶. الگوریتم k-means مینی‌دسته‌ای دارای اینرسی بالاتری نسبت به k-means معمولی است (نمودار سمت چپ)، اما بسیار سریع‌تر عمل می‌کند (نمودار سمت راست)، به‌ویژه زمانی که مقدار k افزایش می‌یابد.

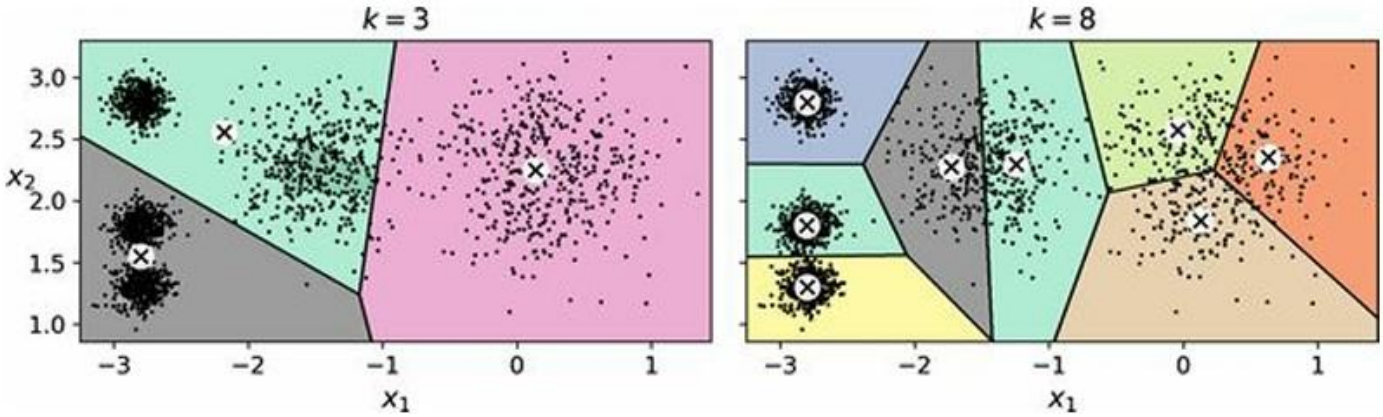
یافتن تعداد بهینه خوشه‌ها

تا اینجا، تعداد خوشه‌ها (k) را برابر ۵ قرار دادیم، چون با نگاه کردن به داده‌ها به‌وضوح مشخص بود که این مقدار مناسب است. اما در حالت کلی، تعیین مقدار مناسب برای k به این سادگی نخواهد بود،

© 2025 Mohsen Zare. All rights reserved.

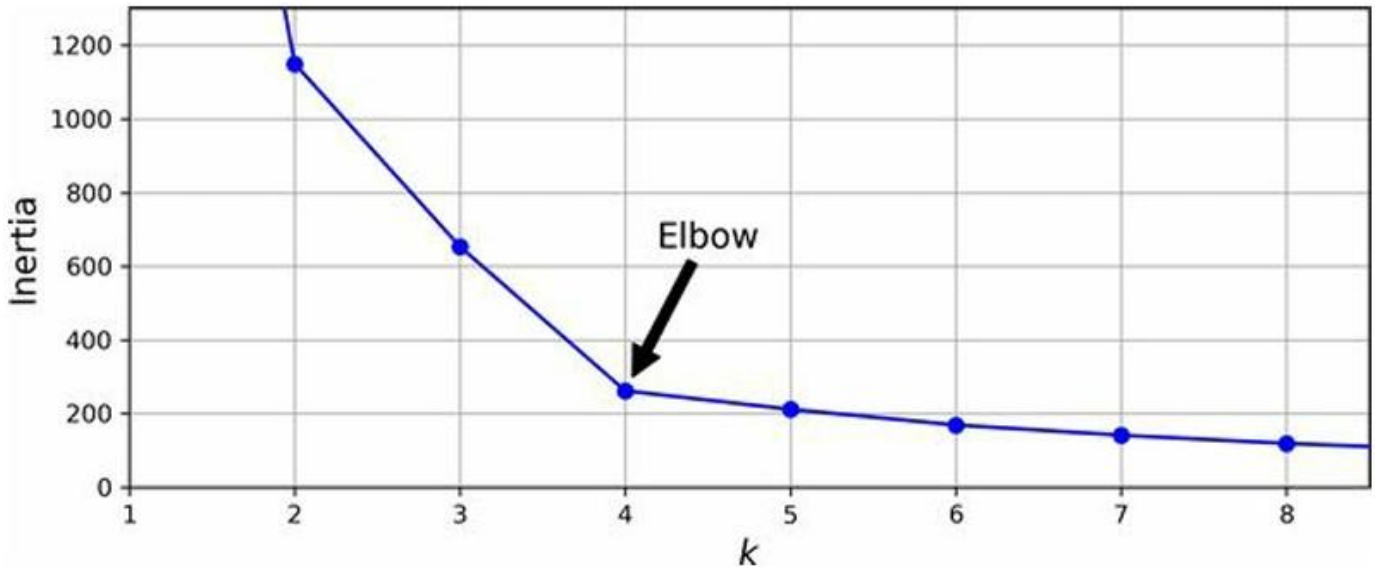
و اگر مقدار اشتباهی برای آن انتخاب شود، نتیجه می‌تواند بسیار ضعیف باشد. همان‌طور که در شکل ۹-۷ مشاهده می‌کنید، برای این مجموعه داده، تنظیم k برابر با ۳ یا ۸ منجر به مدل‌هایی نسبتاً ضعیف می‌شود.

ممکن است با خودتان فکر کنید که می‌توان مدلی با کمترین مقدار اینرسی را انتخاب کرد. اما متأسفانه، این کار به این سادگی نیست. برای مثال، مقدار اینرسی زمانی که $k = 3$ است حدود ۶۵۳٫۲ می‌باشد، که خیلی بیشتر از مقدار آن در $k = 5$ (یعنی ۲۱۱٫۶) است. اما زمانی که $k = 8$ قرار می‌دهیم، اینرسی به عدد نسبتاً کوچکی مانند ۱۱۹٫۱ کاهش می‌یابد. اینرسی معیار مناسبی برای ارزیابی عملکرد هنگام انتخاب مقدار k نیست، زیرا با افزایش k، مقدار اینرسی همواره کاهش می‌یابد. در واقع، هرچه تعداد خوشه‌ها بیشتر باشد، فاصله‌ی هر نمونه تا نزدیک‌ترین مرکز خوشه کمتر می‌شود، و در نتیجه، اینرسی نیز کمتر خواهد بود. برای بررسی این موضوع، بیایید اینرسی را به‌عنوان تابعی از k رسم کنیم. وقتی این کار را انجام می‌دهیم، نمودار معمولاً نقطه‌ای خمیده یا شکسته دارد که به آن «نقطه آرنجی($elbow$)» می‌گویند (نگاه کنید به شکل ۹-۸).



شکل ۹-۷. انتخاب‌های نامناسب برای تعداد خوشه‌ها: وقتی مقدار k خیلی کوچک باشد، خوشه‌های مجزا با یکدیگر ترکیب می‌شوند (سمت چپ)، و وقتی مقدار k خیلی بزرگ باشد، برخی خوشه‌ها به چند بخش تقسیم می‌شوند (سمت راست).

© 2025 Mohsen Zare. All rights reserved.



شکل ۹-۸. نمودار اینرسی (Inertia) بر حسب تعداد خوشه‌ها (k)

همان‌طور که می‌بینید، با افزایش مقدار k تا حدود ۴، اینرسی به‌سرعت کاهش می‌یابد، اما پس از آن، کاهش اینرسی بسیار کندتر می‌شود. این نمودار شکلی شبیه به یک بازو دارد و در مقدار $k = 4$ یک نقطه‌ی زانویی(elbow) دیده می‌شود. بنابراین، اگر اطلاعات بیشتری نداشتیم، ممکن بود فکر کنیم که مقدار ۴ انتخاب مناسبی است: مقادیر کمتر از آن باعث ادغام خوشه‌های جداگانه می‌شوند و بسیار بد هستند، در حالی که مقادیر بیشتر نیز سود چندانی ندارند و فقط ممکن است خوشه‌های کاملاً مناسب را بی‌دلیل به چند قسمت تقسیم کنیم.

این روش برای انتخاب بهترین مقدار خوشه‌ها (k) نسبتاً خام و تقریبی است. یک روش دقیق‌تر (اما پرهزینه‌تر از نظر محاسباتی) استفاده از «امتیازِ سیلوئت ($silhouette\ score$)» است، که برابر است با میانگینِ «ضریبِ سیلوئتِ($silhouette\ coefficient$)» تمام نمونه‌ها. ضریب سیلوئت یک نمونه برابر است با: $(b - a)/max(a, b)$ ، که در آن a میانگین فاصله نمونه تا سایر نمونه‌های همان خوشه (یعنی میانگین فاصله درون‌خوشه‌ای) و b میانگین فاصله تا نزدیک‌ترین خوشه دیگر (یعنی میانگین فاصله تا نمونه‌های نزدیک‌ترین خوشه‌یِ بعدی که به گونه‌ای تعریف شده که b را مینیمم می‌کند، به جز خوشه خود نمونه) است. ضریب سیلوئت می‌تواند مقداری بین ۱- تا ۱+ داشته باشد. ضریبی نزدیک به ۱+ نشان می‌دهد که نمونه کاملاً در داخل خوشه خود قرار دارد و از سایر خوشه‌ها دور است، در حالی‌که ضریبی نزدیک به صفر بیانگر آن است که نمونه به مرز بین خوشه‌ها نزدیک است؛ و در نهایت،

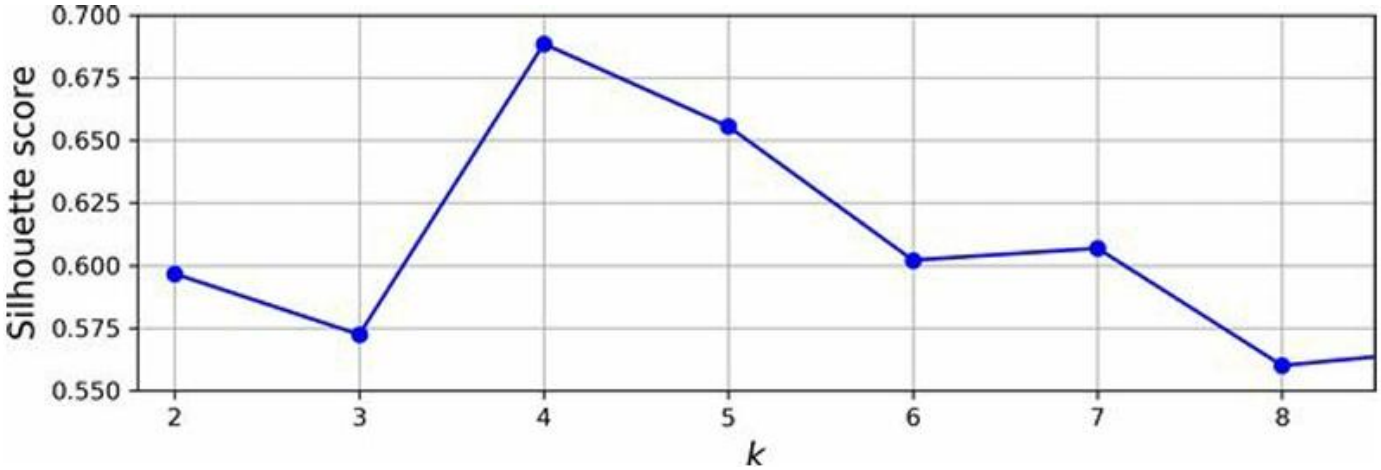
© 2025 Mohsen Zare. All rights reserved.

ضریبی نزدیک به ۱- نشان می‌دهد که ممکن است نمونه به خوشه‌ای اشتباه اختصاص داده شده باشد.

برای محاسبه ضریب یا امتیازِ سیلوئت، می‌توانید از تابع silhouette_score() در Scikit-Learn استفاده کنید، به طوری که تمام نمونه‌های داده و برچسب‌هایی که به آن‌ها اختصاص داده شده را به این تابع بدهید:

```
>>> from sklearn.metrics import silhouette_score
>>> silhouette_score(X, kmeans.labels_)
0.65551764257
```

بیایید امتیازهای سیلوئت را برای تعداد خوشه‌های مختلف مقایسه کنیم (به شکل ۹-۹ مراجعه کنید).



شکل ۹-۹. انتخاب تعداد خوشه‌ها k با استفاده از امتیاز سیلوئت

همان‌طور که می‌بینید، این نمودار نسبت به نمودار قبلی بسیار اطلاعات بیشتری می‌دهد: اگرچه تأیید می‌کند که $k = 4$ انتخاب بسیار خوبی است، اما همچنین نشان می‌دهد که $k = 5$ هم گزینه مناسبی است و خیلی بهتر از $k = 6$ یا $k = 7$ است. این موضوع هنگام مقایسه اینرسی‌ها قابل مشاهده نبود.

یک مصورسازی بسیار آموزنده‌تر زمانی به دست می‌آید که ضریب سیلوئت هر نمونه را بر اساس خوشه‌ای که به آن اختصاص داده شده و مقدارِ ضریب مرتب کنیم و رسم کنیم. این نمودار را «نمودارِ سیلوئت ($silhouette\ diagram$)» می‌نامند (به شکل ۹-۱۰ نگاه کنید). هر خوشه در این نمودار به

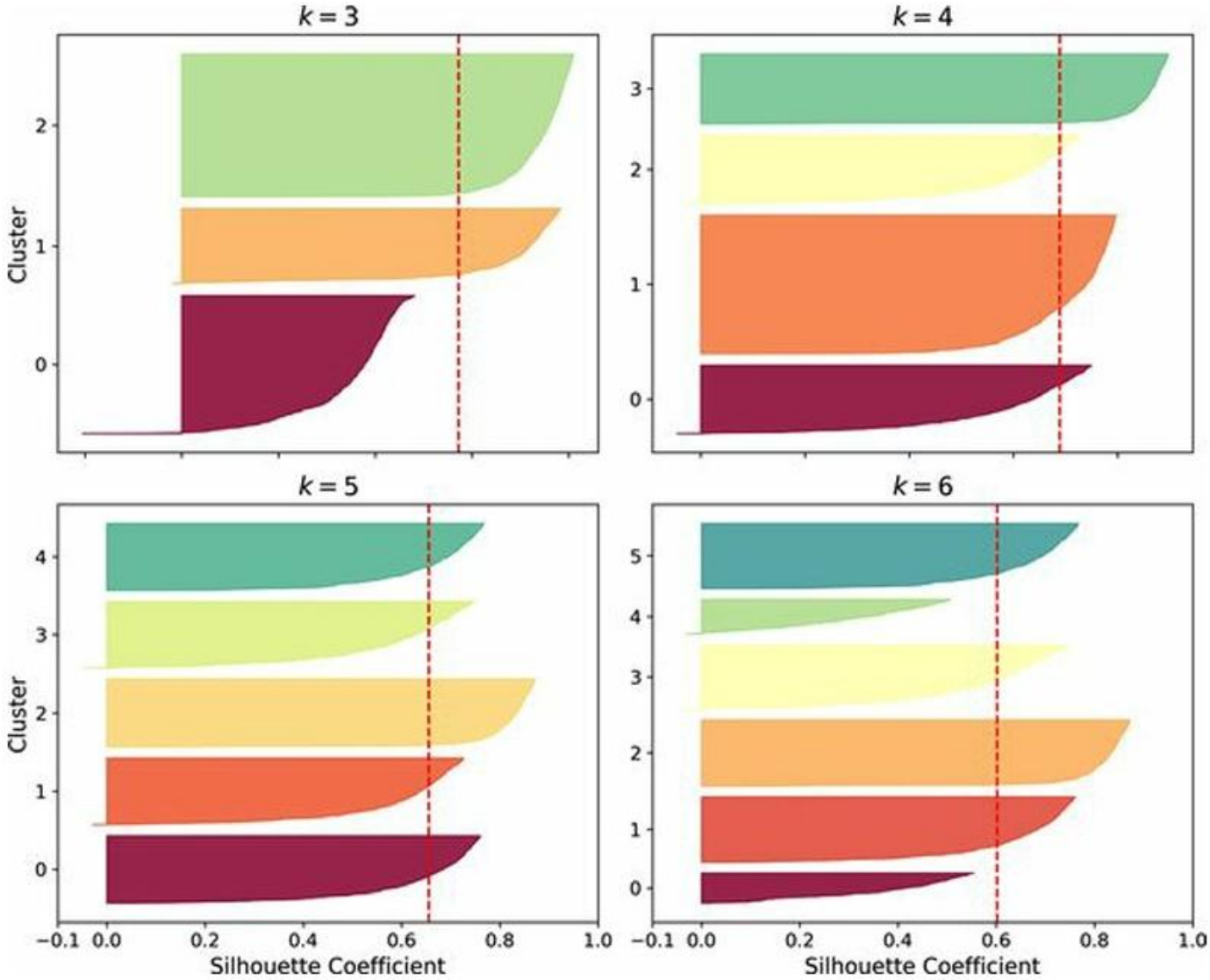
© 2025 Mohsen Zare. All rights reserved.

شکل یک تیغه چاقو نمایش داده می‌شود. ارتفاع شکل نشان‌دهنده تعداد نمونه‌ها در خوشه است و عرض آن نمایانگر ضریب‌های سیلوئت مرتب‌شده‌ی نمونه‌ها در خوشه است (هرچه عرض بیشتر باشد، بهتر است).

خطوط عمودی خطچین نشان‌دهنده میانگین امتیازِ سیلوئت برای هر تعداد خوشه هستند. وقتی بیشتر نمونه‌های یک خوشه ضریب پایین‌تری از این مقدار میانگین دارند (یعنی اگر بسیاری از نمونه‌ها به خطچین نمی‌رسند و در سمت چپ آن پایان می‌یابند)، آن خوشه وضعیت خوبی ندارد، چون یعنی نمونه‌های آن خیلی به خوشه‌های دیگر نزدیک هستند. در اینجا می‌بینیم وقتی k برابر ۳ یا ۶ است، خوشه‌های نامناسبی داریم. اما وقتی k برابر ۴ یا ۵ باشد، خوشه‌ها وضعیت بهتری دارند: بیشتر نمونه‌ها از خطچین عبور می‌کنند و به سمت راست و نزدیک به مقدار ۱٫۰ قرار دارند.

وقتی $k = 4$ باشد، خوشه‌ای که در اندیس ۱ قرار دارد (دومین خوشه از پایین) نسبتاً بزرگ است. اما وقتی k برابر ۵ باشد، همه خوشه‌ها اندازه‌های مشابهی دارند. بنابراین، هرچند امتیاز کلی سیلوئت برای $k = 4$ کمی بیشتر از $k = 5$ است، به نظر می‌رسد استفاده از $k = 5$ انتخاب بهتری باشد تا خوشه‌هایی با اندازه‌های نزدیک به هم داشته باشیم.

© 2025 Mohsen Zare. All rights reserved.



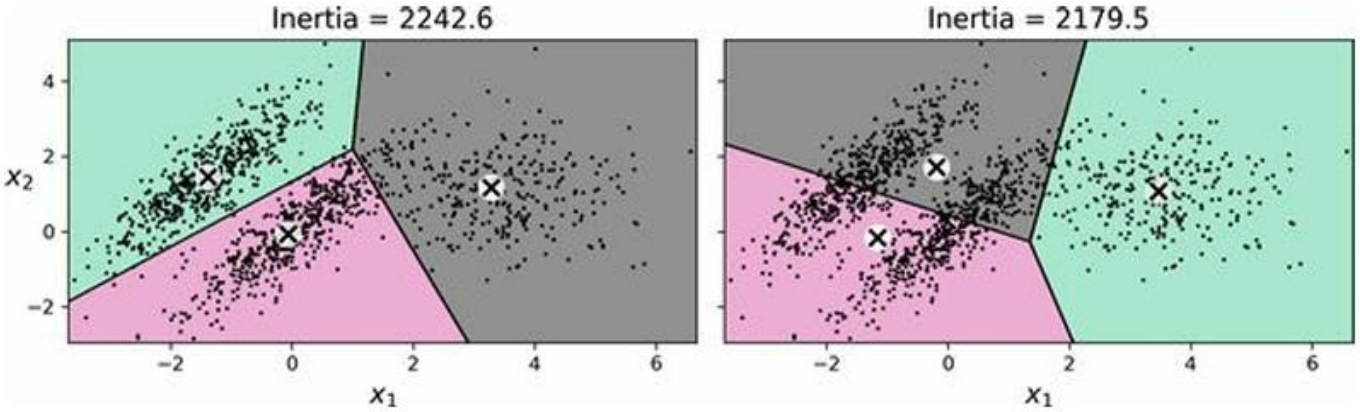
شکل ۹-۱۰. تحلیل نمودارهای سیلوئت برای مقادیر مختلف k

© 2025 Mohsen Zare. All rights reserved.

محدودیت‌های k-means

با وجود مزایای فراوانش، به‌خصوص سرعت و قابلیت مقیاس‌پذیری و گسترش بالا، الگوریتم k-means بی‌نقص نیست. همان‌طور که دیدیم، باید الگوریتم را چند بار اجرا کرد تا از رسیدن به راه‌حل‌های غیربهینه جلوگیری شود، علاوه بر این باید تعداد خوشه‌ها را مشخص کنید که ممکن است کار دشواری باشد. علاوه بر این، k-means زمانی که خوشه‌ها اندازه‌های متفاوت، چگالی‌های مختلف یا شکل‌های غیرکروی داشته باشند، عملکرد مناسبی ندارد. برای مثال، شکل ۹-۱۱ نشان می‌دهد که چگونه k-means یک مجموعه داده شامل سه خوشه بیضی‌شکل با ابعاد، چگالی‌ها و جهت‌گیری‌های متفاوت را خوشه‌بندی می‌کند.

همانطور که می‌بینید، هیچ‌کدام از این دو راه‌حل خوب نیستند. راه‌حل سمت چپ بهتر است، اما هنوز ۲۵% از خوشه میانی را جدا کرده و به خوشه سمت راست اختصاص می‌دهد. راه‌حل سمت راست واقعاً ضعیف است، حتی اگر مقدار اینرسی آن کمتر باشد. بنابراین، بسته به نوع داده‌ها، الگوریتم‌های مختلف خوشه‌بندی ممکن است عملکرد بهتری داشته باشند. در مورد این نوع خوشه‌های بِیضَوی، مدل‌های مخلوطِ گاوسی عملکرد بسیار خوبی دارند.



شکل ۹-۱۱. الگوریتم k-means در خوشه‌بندی صحیح این حباب‌ها یا خوشه‌های بیضوی، ناموفق است

© 2025 Mohsen Zare. All rights reserved.

نکته

قبل از اجرای k-means مهم است که ویژگی‌های ورودی را مقیاس‌بندی کنید (رجوع کنید به فصل ۲)، وگرنه خوشه‌ها ممکن است بسیار کشیده شوند و عملکرد k-means ضعیف خواهد بود. مقیاس‌بندی ویژگی‌ها تضمین نمی‌کند که همه خوشه‌ها کاملاً کروی باشند، اما معمولاً به بهبود عملکرد k-means کمک می‌کند.

حالا بیایید چند روش را بررسی کنیم که چگونه می‌توانیم از خوشه‌بندی بهره ببریم. ما از الگوریتم k-means استفاده خواهیم کرد، اما شما هم می‌توانید با سایر الگوریتم‌های خوشه‌بندی آزمایش کنید.

© 2025 Mohsen Zare. All rights reserved.

استفاده از خوشه‌بندی برای بخش‌بندی تصویر

قطعه‌بندی یا تقسیم‌بندی تصویر (*Image segmentation*) وظیفه‌ی بخش‌بندی یک تصویر به چند بخشِ(segment) مختلف را بر عهده دارد. این کار انواع مختلفی دارد:

- در «بخش‌بندی رنگی (*Color segmentation*)»، پیکسل‌هایی که رنگ مشابهی دارند به یک بخش مشترک اختصاص داده می‌شوند. این روش در بسیاری از کاربردها کافی است. برای مثال، اگر بخواهید تصاویر ماهواره‌ای را برای اندازه‌گیری میزان کل مساحت جنگل در یک منطقه تحلیل کنید، تقسیم‌بندی رنگ می‌تواند کاملاً مناسب باشد.
- در «بخش‌بندی معنایی (*Semantic Segmentation*)»، تمام پیکسل‌هایی که متعلق به یک نوع شیء هستند، به یک بخش اختصاص داده می‌شوند. برای مثال، در سیستم بینایی یک خودروی خودران، تمام پیکسل‌هایی که بخشی از تصویر یک عابر پیاده هستند، ممکن است به بخش «عابر پیاده» اختصاص یابند (یعنی یک بخش وجود دارد که شامل تمام عابرهای پیاده است).
- در «بخش‌بندی نمونه‌ای (*Instance Segmentation*)»، تمام پیکسل‌هایی که متعلق به یک شیء منحصربه‌فرد و مشخص هستند، به یک بخش اختصاص داده می‌شوند. در این حالت، برای هر عابر پیاده یک بخش جداگانه وجود خواهد داشت.

پیشرفته‌ترین روش‌ها در زمینهٔ بخش‌بندی معنایی یا تقسیم‌بندی نمونه‌ای، امروزه با استفاده از معماری‌های پیچیده‌ای مبتنی بر شبکه‌های عصبی کانولوشنی (*convolutional neural networks*) به‌دست می‌آیند (به جلد دوم فصل ۱۴ مراجعه کنید). اما در این فصل، تمرکز ما بر روی مسئله‌ی بسیار ساده‌ترِ بخش‌بندی رنگ با استفاده از الگوریتم k-means خواهد بود.

ابتدا پکیج Pillow را که نسخه‌ی به‌روزشده‌ی کتابخانه‌ی Python Imaging Library (PIL) است، وارد می‌کنیم. از این بسته برای بارگذاری تصویر ladybug.png استفاده خواهیم کرد (تصویر بالا-چپ در شکل ۹-۱۲ را ببینید)، با فرض اینکه این فایل در مسیرِ filepath قرار دارد:

```
>>> import PIL
```

© 2025 Mohsen Zare. All rights reserved.

```
>>> image = np.asarray(PIL.Image.open(filepath))
>>> image.shape
(533, 800, 3)
```

تصویر به‌صورت یک آرایه‌ی سه‌بُعدی نمایش داده می‌شود. اندازه‌ی بُعد اول نشان‌دهنده‌ی ارتفاع تصویر است، بُعد دوم عرض، و بُعد سوم تعداد کانال‌های رنگی است که در اینجا شامل قرمز، سبز و آبی (RGB) می‌شود. به‌عبارت دیگر، برای هر پیکسل یک بردار سه‌بُعدی وجود دارد که شدت رنگ‌های قرمز، سبز و آبی را به‌صورت اعداد صحیح بدون علامت ۸ بیتی بین ۰ تا ۲۵۵ در خود نگه می‌دارد. برخی تصاویر ممکن است کانال‌های کمتری داشته باشند (مانند تصاویر خاکستری که تنها یک کانال دارند)، و برخی دیگر ممکن است کانال‌های بیشتری داشته باشند (مثلاً تصاویری که یک «کانال آلفا (*alpha channel*)» اضافی برای شفافیت دارند، یا تصاویر ماهواره‌ای که معمولاً شامل کانال‌هایی برای فرکانس‌های نوری اضافی مانند مادون قرمز هستند).

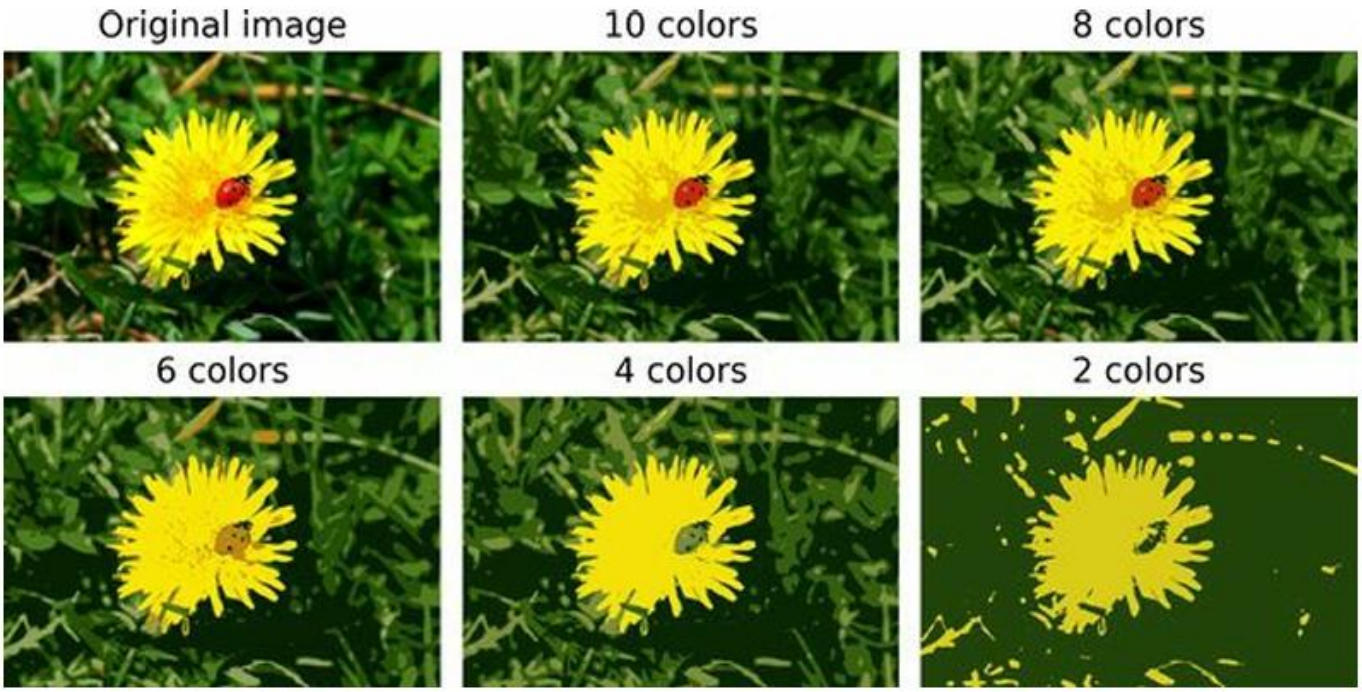
کد زیر آرایه تصویر را بازشکل‌دهی می‌کند(تغییر شکل می‌دهد؛ یا reshape میکند) تا به یک لیست بلند از رنگ‌های RGB تبدیل شود، سپس این رنگ‌ها را با استفاده از الگوریتم k-means و ۸ خوشه دسته‌بندی می‌کند. سپس، آرایه‌ای به نام segmented_img می‌سازد که برای هر پیکسل، نزدیک‌ترین مرکز خوشه را (یعنی میانگین رنگ خوشه‌ای که آن پیکسل در آن قرار دارد) در خود دارد. در نهایت، این آرایه را به شکل اصلی تصویر بازمی‌گرداند(reshape). خط سوم این کد از اندیس‌گذاری پیشرفته‌ی NumPy استفاده می‌کند؛ برای مثال، اگر ۱۰ برچسب اول در kmeans_.labels_ برابر با ۱ باشند، آنگاه ۱۰ رنگ اول در segmented_img برابر با kmeans.cluster_centers_[1] خواهند بود:

```
X = image.reshape(-1, 3)
kmeans = KMeans(n_clusters=8, random_state=42).fit(X)
segmented_img = kmeans.cluster_centers_[kmeans.labels_]
segmented_img = segmented_img.reshape(image.shape)
```

این کد تصویری را تولید می‌کند که در گوشهٔ بالا سمت راست شکل ۹-۱۲ نمایش داده شده است. می‌توانید با تعداد خوشه‌های مختلف آزمایش کنید، همان‌طور که در شکل نشان داده شده است. وقتی از کمتر از هشت خوشه استفاده می‌کنید، متوجه می‌شوید که رنگِ قرمزِ درخشانِ کفشدوزک، خوشه‌ی جداگانه‌ای دریافت نمی‌کند؛ بلکه با رنگ‌های محیط ترکیب می‌شود. دلیل این موضوع این است که الگوریتم k-means تمایل دارد خوشه‌هایی با اندازه‌های نسبتاً مشابه بسازد. از آنجا که کفشدوزک

© 2025 Mohsen Zare. All rights reserved.

کوچک است—بسیار کوچک‌تر از بقیه تصویر—بنابراین حتی با وجود رنگ درخشانش، الگوریتم k-means نتوانسته یک خوشه‌ی اختصاصی برای آن در نظر بگیرد.



شکل ۹-۱۲. بخش‌بندی تصویر با استفاده از الگوریتم k-means و تعدادهای مختلفی از خوشه‌های رنگی (Color Clusters)

خیلی هم سخت نبود، نه؟ حالا بیایید به کاربرد دیگری از خوشه‌بندی نگاه کنیم.

© 2025 Mohsen Zare. All rights reserved.

استفاده از خوشه‌بندی برای یادگیری نیمه‌نظارتی

کاربرد دیگر خوشه‌بندی در یادگیری نیمه‌نظارتی (semi-supervised learning) است؛ زمانی که تعداد زیادی نمونه بدون برچسب و تنها تعداد کمی نمونه برچسب‌دار داریم. در این بخش، از مجموعه‌داده digits استفاده می‌کنیم؛ مجموعه‌ای ساده و شبیه به MNIST که شامل ۱,۷۹۷ تصویر خاکستری ۸×۸ از ارقام ۰ تا ۹ است. ابتدا بیایید این مجموعه داده را بارگذاری و تقسیم کنیم (این مجموعه از پیش به‌طور تصادفی به هم ریخته شده است):

```
from sklearn.datasets import load_digits

X_digits, y_digits = load_digits(return_X_y=True)
X_train, y_train = X_digits[:1400], y_digits[:1400]
X_test, y_test = X_digits[1400:], y_digits[1400:]
```

ما وانمود می‌کنیم که تنها برای ۵۰ نمونه، برچسب داریم. برای به‌دست‌آوردن یک معیار پایه از عملکرد (baseline)، ابتدا یک مدل رگرسیون لجستیک را فقط روی همین ۵۰ نمونه‌ی برچسب‌دار آموزش می‌دهیم:

```
from sklearn.linear_model import LogisticRegression

n_labeled = 50
log_reg = LogisticRegression(max_iter=10_000)
log_reg.fit(X_train[:n_labeled], y_train[:n_labeled])
```

سپس می‌توانیم دقت (accuracy) این مدل را روی مجموعه‌ی آزمون اندازه‌گیری کنیم (توجه داشته باشید که مجموعه‌ی آزمون باید دارای برچسب باشد):

```
>>> log_reg.score(X_test, y_test)
0.7481108312342569
```

دقت مدل فقط ۷۴٫۸ % است که زیاد خوب نیست. در واقع اگر مدل را روی کل مجموعه آموزش

© 2025 Mohsen Zare. All rights reserved.

تمرین دهید، دقت آن حدود ۹۰٫۷ %خواهد بود. حالا ببینیم چطور می‌توان بهتر عمل کرد. اول، مجموعه آموزش را به ۵۰ خوشه تقسیم می‌کنیم. سپس برای هر خوشه، تصویری که به مرکز خوشه نزدیک‌تر است را پیدا می‌کنیم. این تصاویر را «تصاویر شاخص یا نماینده (*representative images*)»— تصویری که ویژگی‌های اصلی یا مشترک یک دسته (class) یا خوشه (cluster) از تصاویر را به‌طور خلاصه نشان می‌دهد. مثلاً در خوشه‌بندی تصاویر، نماینده‌ترین تصویر هر خوشه انتخاب می‌شود— می‌نامیم:

```
k = 50
kmeans = KMeans(n_clusters=k, random_state=42)
X_digits_dist = kmeans.fit_transform(X_train)
representative_digit_idx = np.argmin(X_digits_dist, axis=0)
X_representative_digits = X_train[representative_digit_idx]
```

شکل ۹-۱۳، ۵۰ تصویر نمونه را نشان می‌دهد.

شکل ۹-۱۳. پنجاه تصویر نماینده از اعداد (یک تصویر برای هر خوشه)

بیایید هر تصویر را ببینیم و به‌صورت دستی برچسب‌گذاری کنیم:

```
y_representative_digits = np.array([1, 3, 6, 0, 7, 9, 2, ..., 5, 1, 9, 9, 3, 7])
```

حالا مجموعه‌داده‌ای داریم با فقط ۵۰ نمونه برچسب‌خورده، اما به جای اینکه نمونه‌های تصادفی باشند، هر کدام تصویرِ نماینده‌ای از خوشه خودش است. بیاییم ببینیم آیا عملکرد بهتر شده است یا نه:

```
>>> log_reg = LogisticRegression(max_iter=10_000)
```

© 2025 Mohsen Zare. All rights reserved.

```
>>> log_reg.fit(X_representative_digits, y_representative_digits)
>>> log_reg.score(X_test, y_test)
0.84886649874
```

عجب! دقت مدل از ۷۴٫۸ % به ۸۴٫۹ % افزایش یافت، در حالی که هنوز فقط روی ۵۰ نمونه آموزش می‌دهیم. چون برچسب‌زنی نمونه‌ها اغلب پرهزینه و دشوار است، به‌ویژه وقتی باید به صورت دستی و توسط متخصصان انجام شود، بهتر است نمونه‌های نماینده‌ی هر خوشه را برچسب‌گذاری کنیم تا نمونه‌هایِ تصادفی.

اما شاید بتوانیم یک قدم جلوتر برویم: فرض کنید برچسب‌ها را به همه نمونه‌های دیگر در همان خوشه منتقل(منتشر) کنیم. این کار را «انتشار برچسب (*label propagation*)»—انتشار برچسب‌های تعداد کمی داده‌ی برچسب‌دار روی کل داده‌ها (بر اساس شباهت‌ها یا روابط گرافی)—می‌نامند:

```
y_train_propagated = np.empty(len(X_train), dtype=np.int64)
for i in range(k):
  y_train_propagated[kmeans.labels_ == i] = y_representative_digits[i]
```

حالا بیایید دوباره مدل را آموزش دهیم و عملکرد آن را بررسی کنیم:

```
>>> log_reg = LogisticRegression()
>>> log_reg.fit(X_train, y_train_propagated)
>>> log_reg.score(X_test, y_test)
0.89420654911
```

یک افزایش دقت چشمگیر دیگر هم به دست آمد! حالا بیایید ببینیم آیا می‌توانیم با حذف ۱% از نمونه‌هایی که از مرکز خوشه‌ی خود بیشترین فاصله را دارند، عملکرد را باز هم بهبود دهیم؟ این کار باعث می‌شود برخی داده‌های پرت حذف شوند. کد زیر ابتدا فاصله‌ی هر نمونه را تا نزدیک‌ترین مرکز خوشه محاسبه می‌کند، سپس برای هر خوشه، ۱% از بزرگ‌ترین فاصله‌ها را با مقدار ۱- علامت‌گذاری می‌کند. در نهایت، مجموعه‌ای جدید بدون این نمونه‌های علامت‌خورده ایجاد می‌کند:

```
percentile_closest = 99

X_cluster_dist = X_digits_dist[np.arange(len(X_train)), kmeans.labels_]
for i in range(k):
```

© 2025 Mohsen Zare. All rights reserved.

```
  in_cluster = (kmeans.labels_ == i)
  cluster_dist = X_cluster_dist[in_cluster]
  cutoff_distance = np.percentile(cluster_dist, percentile_closest)
  above_cutoff = (X_cluster_dist > cutoff_distance)
  X_cluster_dist[in_cluster & above_cutoff] = -1

partially_propagated = (X_cluster_dist != -1)
X_train_partially_propagated = X_train[partially_propagated]
y_train_partially_propagated = y_train_propagated[partially_propagated
```

حالا بیایید مدل را دوباره روی این مجموعه‌داده که برچسب‌ها به‌صورت جزئی در آن منتشر شده‌اند (partially propagated) آموزش دهیم و ببینیم چه دقتی به دست می‌آید:

```
>>> log_reg = LogisticRegression(max_iter=10_000)
>>> log_reg.fit(X_train_partially_propagated, y_train_partially_propagated)
>>> log_reg.score(X_test, y_test)
0.9093198992443325
```

عالیه! با تنها ۵۰ نمونه برچسب‌خورده (یعنی به‌طور میانگین فقط ۵ نمونه برای هر کلاس!) به دقت ۹۰٫۹% رسیدیم، که حتی کمی بالاتر از عملکردی است که روی مجموعه‌داده‌ی کاملِ برچسب‌خورده‌ی ارقام به دست آوردیم (۹۰٫۷ %). این نتیجه تا حدی به خاطر حذف برخی داده‌های پرت و تا حدی هم به این دلیل است که برچسب‌های منتشرشده (propagated labels) واقعاً کیفیت خوبی دارند—دقت آن‌ها حدود ۹۷٫۵ % است، همان‌طور که کد زیر نشان می‌دهد:

```
>>> (y_train_partially_propagated == y_train[partially_propagated]).mean()
0.9755555555555555
```

© 2025 Mohsen Zare. All rights reserved.

نکته

کتابخانه‌ی Scikit-Learn همچنین دو کلاس برای انتشار خودکار برچسب‌ها ارائه می‌دهد: LabelSpreading و LabelPropagation در پکیج sklearn.semi_supervised. هر دو کلاس یک ماتریس شباهت بین تمام نمونه‌ها ایجاد می‌کنند و سپس به‌صورت تکراری، برچسب‌ها را از نمونه‌های برچسب‌خورده به نمونه‌های مشابه بدون برچسب منتشر یا منتقل می‌کنند. همچنین در همان بسته، کلاسی کاملاً متفاوت به نام SelfTrainingClassifier وجود دارد: شما یک مدل پایه (مثلاً RandomForestClassifier) را به آن می‌دهید و ابتدا آن را روی نمونه‌های برچسب‌خورده آموزش می‌دهد. سپس از آن برای پیش‌بینی برچسب‌های نمونه‌های بدون برچسب استفاده می‌کند. در ادامه، مجموعه‌ی آموزش را با برچسب‌هایی که از آن‌ها بیشترین اطمینان را دارد به‌روزرسانی می‌کند، و این فرآیند آموزش و برچسب‌گذاری را تکرار می‌کند تا دیگر نتواند برچسبی اضافه کند. این تکنیک‌ها معجزه‌آسا نیستند، اما گاهی اوقات می‌توانند کمی عملکرد مدل شما را بهبود دهند.

یادگیری فعال

برای ادامه‌ی بهبود مدل و مجموعه‌ی آموزش، گام بعدی می‌تواند انجام چند دور «یادگیری فعال (*Active Learning*)» باشد؛ حالتی که در آن یک کارشناس انسانی با الگوریتم یادگیری تعامل دارد و برای نمونه‌های خاصی که الگوریتم درخواست می‌کند، برچسب تعیین می‌کند. استراتژی‌های مختلفی برای یادگیری فعال وجود دارد، اما یکی از رایج‌ترین آن‌ها «نمونه‌گیری بر اساس عدم قطعیت (*Uncertainty Sampling*)» نام دارد. در ادامه، نحوه‌ی عملکرد آن توضیح داده شده است:

۱. مدل روی نمونه‌های برچسب‌خورده‌ای که تا این لحظه جمع‌آوری شده‌اند آموزش می‌بیند، و سپس از این مدل برای پیش‌بینی برچسب تمام نمونه‌های بدون برچسب استفاده می‌شود.

۲. نمونه‌هایی که مدل بیشترین عدم قطعیت(عدم اطمینان) را نسبت به آن‌ها دارد (یعنی جایی که احتمال پیش‌بینی‌شده کمترین مقدار را دارد) به کارشناس داده می‌شوند تا آن‌ها را برچسب‌گذاری کند.

۳. این فرایند را تکرار می‌کنید تا زمانی که بهبود عملکرد دیگر ارزش صرف زمان برای برچسب‌گذاری را نداشته باشد.

© 2025 Mohsen Zare. All rights reserved.

سایر استراتژی‌های یادگیری فعال شامل برچسب‌گذاری نمونه‌هایی است که باعث بزرگ‌ترین تغییر در مدل می‌شوند یا بیشترین کاهش در خطای اعتبارسنجی مدل را به همراه دارند، یا نمونه‌هایی که مدل‌های مختلف روی آن‌ها اختلاف نظر دارند (مثلاً یک SVM و یک جنگل تصادفی).

قبل از اینکه به مدل‌های مخلوط گوسی بپردازیم، بیایید نگاهی به الگوریتم خوشه‌بندی محبوب دیگری به نام DBSCAN بیندازیم که رویکردی کاملاً متفاوت بر پایه تخمین چگالی محلی ارائه می‌دهد. این رویکرد به الگوریتم اجازه می‌دهد تا خوشه‌هایی با اشکال دلخواه را شناسایی کند.

© 2025 Mohsen Zare. All rights reserved.

DBSCAN

الگوریتم «خوشه‌بندی فضایی مبتنی بر چگالی همراه با نویز یا به اختصار DBSCAN (*density-based spatial clustering of applications with noise*)»، خوشه‌ها را به‌صورت نواحی پیوسته‌ای با چگالی بالا تعریف می‌کند. نحوه عملکرد آن به این صورت است:

- برای هر نمونه، الگوریتم تعداد نمونه‌هایی را که در فاصله‌ای کمتر از ε (اپسیلون) از آن قرار دارند می‌شمارد. این ناحیه به عنوان «همسایگی ε (*ε-neighborhood*)» آن نمونه شناخته می‌شود.
- اگر یک نمونه دست‌کم به اندازه‌ی min_samples نمونه (شامل خودش) در همسایگی ε خودش داشته باشد، آنگاه به‌عنوان یک «نمونه‌ی مرکزی(*core instance*)» در نظر گرفته می‌شود. به بیان دیگر، نمونه‌های مرکزی آن‌هایی هستند که در نواحی متراکم قرار دارند.
- تمام نمونه‌هایی که در همسایگی یک نمونه‌ی مرکزی قرار دارند، به همان خوشه تعلق دارند. این همسایگی ممکن است شامل نمونه‌های مرکزی دیگری نیز باشد؛ بنابراین، یک دنباله‌ی بلند از نمونه‌های مرکزی مجاور می‌تواند یک خوشه‌ی واحد را تشکیل دهد.
- هر نمونه‌ای که یک نمونه‌ی مرکزی نیست و در همسایگی‌اش نیز نمونه‌ی مرکزی وجود ندارد، به‌عنوان یک ناهنجاری (anomaly) در نظر گرفته می‌شود.

این الگوریتم زمانی به‌خوبی عمل می‌کند که همه‌ی خوشه‌ها توسط نواحی با چگالی پایین از هم جدا شده باشند. کلاس DBSCAN در کتابخانه‌ی Scikit-Learn به سادگی قابل استفاده است، همان‌طور که انتظار می‌رود. بیایید آن را روی مجموعه داده moons که در فصل ۵ معرفی شد، آزمایش کنیم:

```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=1000, noise=0.05)
dbscan = DBSCAN(eps=0.05, min_samples=5)
dbscan.fit(X)
```

برچسب‌های تمام نمونه‌ها اکنون در متغیر نمونه instance_labels در دسترس هستند:

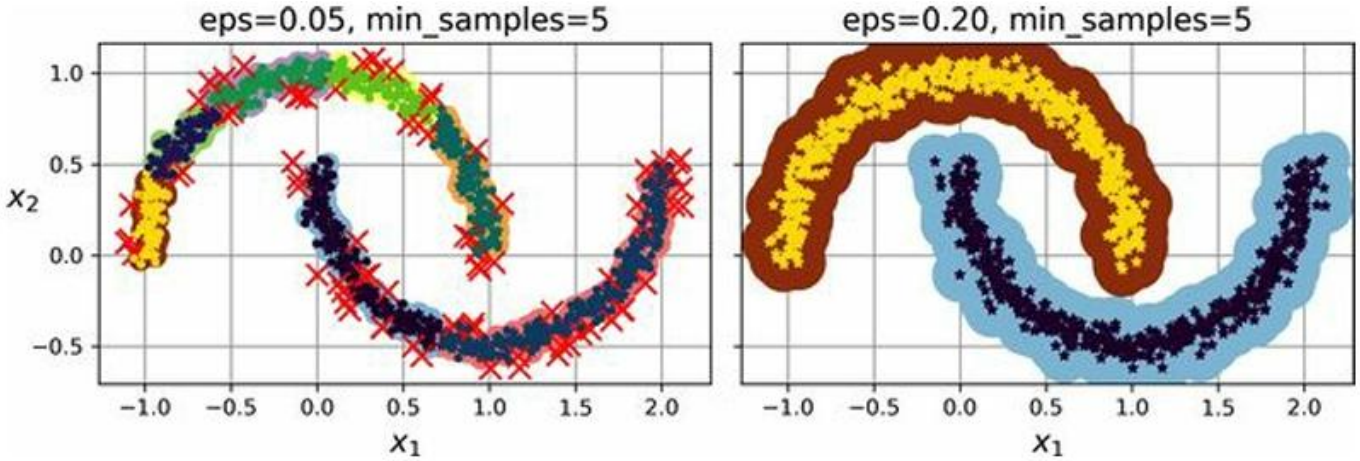
© 2025 Mohsen Zare. All rights reserved.

```
>>> dbscan.labels_
array([ 0,  2, -1, -1,  1,  0,  0,  0,  2,  5, [...], 3,  3,  4,  2,  6,  3])
```

توجه داشته باشید که برخی نمونه‌ها دارای شاخصِ خوشه‌ی برابر با ۱- هستند، که به این معنی است که الگوریتم آن‌ها را به‌عنوان داده‌های ناهنجار(anomalies) در نظر گرفته است. شاخص‌های نمونه‌های مرکزی در متغیر نمونه core_sample_indices_ قرار دارند، و خودِ نمونه‌های مرکزی نیز در متغیر components_ موجود هستند:

```
>>> dbscan.core_sample_indices_
array([  0,   4,   5,   6,   7,   8,  10,  11, [...], 993, 995, 997, 998, 999])
>>> dbscan.components_
array([[-0.02137124,  0.40618608],
       [-0.84192557,  0.53058695],
       [...],
       [ 0.79419406,  0.60777171]])
```

این خوشه‌بندی در نمودار سمت چپ شکل ۹-۱۴ نشان داده شده است. همان‌طور که می‌بینید، الگوریتم تعداد قابل توجهی از نمونه‌ها را به‌عنوان ناهنجار شناسایی کرده و هفت خوشه متفاوت تشخیص داده است. چه ناامیدکننده! خوشبختانه، اگر شعاع همسایگی هر نمونه را با افزایش مقدار eps به ۰٫۲ گسترش دهیم، خوشه‌بندی حاصل در سمت راست نمودار دیده می‌شود که کاملاً مناسب به‌نظر می‌رسد. بیایید با همین مدل ادامه دهیم:



شکل ۹-۱۴. خوشه‌بندی DBSCAN با استفاده از دو شعاع همسایگی متفاوت

© 2025 Mohsen Zare. All rights reserved.

نکته جالب اینجاست که کلاس DBSCAN، متد predict() ندارد، هرچند که دارای متد fit_predict() است. به عبارت دیگر، این مدل نمی‌تواند پیش‌بینی کند که یک نمونه جدید به کدام خوشه تعلق دارد. این تصمیم به این دلیل گرفته شده است که الگوریتم‌های مختلف طبقه‌بندی می‌توانند برای مسائل مختلف مناسب‌تر باشند، بنابراین نویسندگان تصمیم گرفتند انتخاب الگوریتم طبقه‌بندی را به عهده‌ی کاربر بگذارند. البته پیاده‌سازی چنین قابلیتی کار سختی نیست. برای مثال، می‌توان یک مدل KNeighborsClassifier آموزش داد:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=50)
knn.fit(dbscan.components_, dbscan.labels_[dbscan.core_sample_indices_])
```

اکنون، با داشتن چند نمونه‌ی جدید، می‌توانیم پیش‌بینی کنیم که احتمالاً به کدام خوشه تعلق دارند، و حتی می‌توانیم احتمال تعلق هر نمونه به هر خوشه را نیز تخمین بزنیم:

```
>>> X_new = np.array([[-0.5, 0], [0, 0.5], [1, -0.1], [2, 1]])
>>> knn.predict(X_new)
array([1, 0, 1, 0])
>>> knn.predict_proba(X_new)
array([[0.18, 0.82],
       [1.  , 0.  ],
       [0.12, 0.88],
       [1.  , 0.  ]])
```

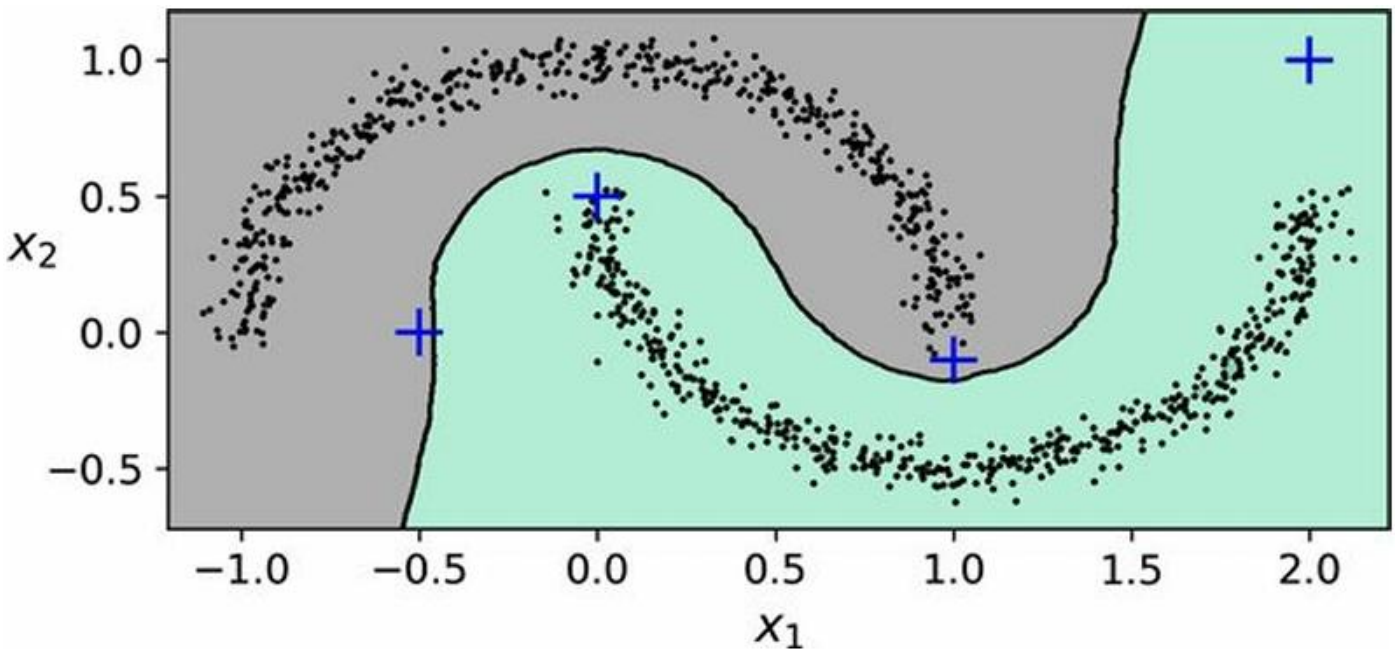
توجه داشته باشید که ما این دسته‌بند را تنها با نمونه‌های مرکزی آموزش دادیم، اما می‌توانستیم آن را با همه‌ی نمونه‌ها، یا با همه‌ی نمونه‌ها به‌جز نمونه‌های ناهنجار آموزش دهیم. این انتخاب بستگی به مسئله‌ی نهایی دارد.

مرز تصمیم‌گیری در شکل ۹-۱۵ نمایش داده شده است (علامت ضربدر نشان‌دهنده‌ی چهار نمونه در X_new است). توجه کنید که چون در مجموعه‌ی آموزش هیچ ناهنجاری‌ای وجود ندارد، دسته‌بند همیشه یک خوشه را انتخاب می‌کند، حتی زمانی که آن خوشه بسیار دور باشد. با این حال، افزودن یک فاصله‌ی حداکثری کار نسبتاً ساده‌ای است؛ در این صورت، دو نمونه‌ای که از هر دو خوشه فاصله‌ی

© 2025 Mohsen Zare. All rights reserved.

زیادی دارند به عنوان ناهنجار طبقه‌بندی می‌شوند. برای انجام این کار می‌توان از متد kneighbors() در KNeighborsClassifier استفاده کرد. این متد با دریافت یک مجموعه از نمونه‌ها، فاصله‌ها و اندیس‌های k همسایه‌ی نزدیک‌تر در مجموعه‌ی آموزش را بازمی‌گرداند (دو ماتریس، هرکدام با k ستون):

```
>>> y_dist, y_pred_idx = knn.kneighbors(X_new, n_neighbors=1)
>>> y_pred = dbscan.labels_[dbscan.core_sample_indices_][y_pred_idx]
>>> y_pred[y_dist > 0.2] = -1
>>> y_pred.ravel()
array([-1,  0,  1, -1])
```



شکل ۹-۱۵. مرز تصمیم‌گیری بین دو خوشه

به طور خلاصه، DBSCAN الگوریتمی بسیار ساده اما قدرتمند است که قادر است هر تعداد خوشه با هر شکلی را شناسایی کند. این الگوریتم نسبت به نقاط پرت مقاوم است و فقط دو هایپرپارامتر دارد (eps و min_samples). با این حال، اگر چگالی خوشه‌ها به طور قابل توجهی متفاوت باشد یا اگر اطراف برخی خوشه‌ها منطقه‌ای با چگالی پایینِ کافی وجود نداشته باشد، DBSCAN ممکن است در شناسایی دقیق همه‌ی خوشه‌ها مشکل داشته باشد. همچنین، پیچیدگی محاسباتی آن تقریباً برابر با $O(m^2n)$ است، بنابراین به خوبی روی مجموعه‌داده‌های بزرگ مقیاس‌پذیر نیست.

© 2025 Mohsen Zare. All rights reserved.

نکته

ممکن است بخواهید الگوریتم *hierarchical DBSCAN* (یا HDBSCAN) را هم امتحان کنید که در پروژه scikit-learn-contrib پیاده‌سازی شده است، زیرا معمولاً در یافتن خوشه‌هایی با چگالی‌های مختلف بهتر از DBSCAN عمل می‌کند.

س

© 2025 Mohsen Zare. All rights reserved.

سایر الگوریتم‌های خوشه‌بندی

کتابخانه Scikit-Learn الگوریتم‌های خوشه‌بندی دیگری را نیز پیاده‌سازی کرده که بد نیست نگاهی به آن‌ها بیندازید. در اینجا امکان بررسی دقیق همه آن‌ها وجود ندارد، اما در ادامه مروری کوتاه بر برخی از آن‌ها آمده است:

خوشه‌بندی تجمعی(*Agglomerative clustering*)

در این روش، یک ساختار سلسله‌مراتبی از خوشه‌ها از پایین به بالا ساخته می‌شود. آن را مانند حباب‌های کوچکی تصور کنید که روی سطح آب شناورند و به‌تدریج به هم می‌چسبند تا در نهایت یک گروه بزرگ از حباب‌ها تشکیل شود. به‌طور مشابه، در هر تکرار از الگوریتم خوشه‌بندی تجمیعی(پیوندی)، نزدیک‌ترین جفت خوشه‌ها به هم متصل می‌شوند (در ابتدا، هر نمونه مجزا خودش یک خوشه مستقل است). اگر این فرایند را به‌صورت درختی رسم کنید که در آن برای هر جفت خوشه‌ای که با هم ترکیب شده‌اند یک شاخه وجود دارد، در نهایت به یک درخت باینری از خوشه‌ها (binary tree of clusters) خواهید رسید، که برگ‌های آن همان نمونه‌های مجزایِ اولیه هستند. این روش قادر است خوشه‌هایی با شکل‌های مختلف را تشخیص دهد؛ همچنین به‌جای اینکه شما را مجبور به انتخاب یک مقیاس مشخص برای خوشه‌ها کند، یک درخت خوشه‌ای منعطف و اطلاعاتی تولید می‌کند. این درخت را می‌توان با هر نوع *فاصله‌ی زوجی* (pairwise distance)به کار برد. اگر یک ماتریس اتصال فراهم کنید، این الگوریتم می‌تواند به‌خوبی برای مجموعه داده‌های بزرگ مقیاس‌پذیر باشد. این ماتریس، یک ماتریس تُنُک (sparse) به ابعاد $m \times m$ است که مشخص می‌کند کدام جفت از نمونه‌ها همسایه هستند (برای مثال، می‌توان آن را با تابع sklearn.neighbors.kneighbors_graph() تولید کرد). اما اگر ماتریس اتصال فراهم نشود، این الگوریتم برای مجموعه داده‌های بزرگ عملکرد خوبی نخواهد داشت.

© 2025 Mohsen Zare. All rights reserved.

BIRCH

الگوریتم «کاهش و خوشه‌بندی متوازن مرحله‌ای با استفاده از سلسله مراتب» یا به اختصار BIRCH (مخفف *Balanced Iterative Reducing and Clustering using Hierarchies*) به‌طور خاص برای مجموعه داده‌های بسیار بزرگ طراحی شده است. این الگوریتم می‌تواند سریع‌تر از k-means دسته‌ای (batch k-means) عمل کند و درعین‌حال نتایجی مشابه ارائه دهد، به شرط آن‌که تعداد ویژگی‌ها زیاد نباشد (کمتر از ۲۰ ویژگی). در مرحله‌ی آموزش، BIRCH یک ساختار درختی می‌سازد که تنها اطلاعات لازم را ذخیره می‌کند تا بتواند به‌سرعت نمونه‌های جدید را به خوشه‌ای اختصاص دهد، بدون اینکه نیاز به ذخیره‌ی همه‌ی نمونه‌ها در درخت باشد. این روش به الگوریتم اجازه می‌دهد که با استفاده از حافظه‌ی محدود، مجموعه داده‌های عظیم را مدیریت کند.

Mean-shift

این الگوریتم با قرار دادن یک دایره به مرکزیت هر نمونه آغاز می‌شود؛ سپس برای هر دایره، میانگین تمام نمونه‌هایی را که درون آن قرار دارند محاسبه کرده و دایره را طوری جابه‌جا می‌کند که مرکز آن روی این میانگین قرار گیرد. در ادامه، این مرحله‌ی جابه‌جاییِ بر اساس میانگین را تکرار می‌کند تا زمانی که همه‌ی دایره‌ها از حرکت بایستند (یعنی زمانی که هر دایره دقیقاً روی میانگین نمونه‌های درون خودش قرار گیرد). الگوریتم میانگینِ‌شیفت یا میانگینِ‌جابجایی (*Mean-Shift*) دایره‌ها را به سمت نواحی با چگالی بالاتر حرکت می‌دهد تا زمانی که هرکدام به یک ماکسیمُم محلی چگالی برسند. در نهایت، تمام نمونه‌هایی که دایره‌هایشان در یک نقطه (یا نقطه‌ای به‌اندازه‌ی کافی نزدیک) پایدار شده‌اند، به یک خوشه نسبت داده می‌شوند. الگوریتم میانگینِ‌جابجایی، ویژگی‌های مشترکی با DBSCAN دارد؛ مثلاً می‌تواند هر تعداد خوشه با هر شکلی را پیدا کند، تنها یک هایپرپارامتر دارد (یعنی شعاع دایره‌ها که «پهنای باند (*bandwidth*)» نامیده می‌شود)، و بر پایه‌ی تخمین چگالی محلی عمل می‌کند. اما برخلاف DBSCAN، میانگینِ‌جابجایی تمایل دارد خوشه‌هایی را که درون خود تغییرات چگالی دارند، به بخش‌های کوچک‌تری تقسیم کند. متأسفانه پیچیدگی محاسباتی آن برابر با $O(m^2 n)$ است؛ بنابراین برای مجموعه داده‌های بزرگ مناسب نیست.

© 2025 Mohsen Zare. All rights reserved.

خوشه‌بندی انتشار وابستگی(*Affinity propagation*)

در این الگوریتم، نمونه‌ها به‌طور مکرر با یکدیگر پیام رد و بدل می‌کنند تا زمانی که هر نمونه، یک نمونه‌ی دیگر (یا خودش) را برای نمایندگی خود انتخاب کند. این نمونه‌های انتخاب‌شده را «نمونه‌های نماینده(*exemplars*)» می‌نامند. هر نماینده به همراه تمام نمونه‌هایی که او را انتخاب کرده‌اند، یک خوشه را تشکیل می‌دهند. در سیاست دنیای واقعی، معمولاً ترجیح می‌دهید به نامزدی رأی بدهید که دیدگاه‌هایش به دیدگاه‌های شما نزدیک است، اما در عین حال می‌خواهید او در انتخابات پیروز شود؛ بنابراین ممکن است کسی را انتخاب کنید که کاملاً با او هم‌عقیده نیستید، اما محبوبیت بیشتری دارد. معمولاً محبوبیت را از طریق نظرسنجی‌ها ارزیابی می‌کنید. الگوریتم انتشارِ وابستگی (Affinity Propagation) به روشی مشابه عمل می‌کند و معمولاً نماینده‌هایی را انتخاب می‌کند که نزدیک به مرکز خوشه‌ها قرار دارند، شبیه به الگوریتم k-means. اما بر خلاف k-means، نیازی به انتخاب تعداد خوشه‌ها پیش از اجرا ندارید؛ این تعداد در طی فرایند آموزش تعیین می‌شود. علاوه بر این، انتشارِ وابستگی می‌تواند به خوبی با خوشه‌هایی با اندازه‌های مختلف کنار بیاید. متأسفانه، این الگوریتم دارای پیچیدگی محاسباتی $O(m^2)$ است، بنابراین برای مجموعه داده‌های بزرگ مناسب نیست.

خوشه‌بندی طیفی(*Spectral clustering*)

این الگوریتم ماتریس شباهت بین نمونه‌ها را می‌گیرد و یک جاسازی (embedding) با ابعاد پایین‌تر از آن ایجاد می‌کند (یعنی بعد ماتریس را کاهش می‌دهد)، سپس از یک الگوریتم خوشه‌بندی دیگر در این فضای با بعد پایین‌تر استفاده می‌کند (پیاده‌سازی Scikit-Learn از k-means استفاده می‌کند). خوشه‌بندی طیفی قادر است ساختارهای پیچیده خوشه‌ها را تشخیص دهد و همچنین می‌تواند برای تقسیم گراف‌ها (مثلاً شناسایی خوشه‌های دوستان در یک شبکه اجتماعی) به کار رود. این الگوریتم برای تعداد زیاد نمونه‌ها مقیاس‌پذیری خوبی ندارد و زمانی که خوشه‌ها اندازه‌های بسیار متفاوتی داشته باشند، عملکرد خوبی از خود نشان نمی‌دهد.

حالا بیایید به مدل‌های مخلوط گاوسی بپردازیم که می‌توان از آن‌ها برای برآورد چگالی، خوشه‌بندی و تشخیص ناهنجاری استفاده کرد.

© 2025 Mohsen Zare. All rights reserved.

مخلوط‌هایِ گاوسی

«مدل مخلوطِ گاوسی (GMM: *Gaussian mixture model*)» یک مدل احتمالاتی است که فرض می‌کند داده‌ها از ترکیبی از چند توزیع گاوسی تولید شده‌اند که پارامترهای آن‌ها ناشناخته است. همه داده‌هایی که از یک توزیع گاوسی تولید شده‌اند، یک خوشه را تشکیل می‌دهند که معمولاً به شکل یک بِیضی دیده می‌شود. هر خوشه می‌تواند شکل، اندازه، چگالی و جهت‌گیری بیضوی متفاوتی داشته باشد، درست مانند شکل ۹-۱۱. زمانی که یک داده را مشاهده می‌کنید، می‌دانید که از یکی از این توزیع‌های گاوسی تولید شده است، اما نمی‌دانید کدام توزیع و پارامترهای آن‌ها چیست.

چندین نسخه متفاوت از مدل مخلوط گاوسی (GMM) وجود دارد. در ساده‌ترین نسخه که در کلاس GaussianMixture پیاده‌سازی شده است، باید تعداد k توزیعِ گاوسی را از قبل بدانید. در این نسخه، فرض می‌شود که داده‌های مجموعه‌ی $\mathbf{X}$ از طریق فرآیند احتمالاتی زیر تولید شده‌اند:

- برای هر نمونه، یکی از k-خوشه به‌صورت تصادفی انتخاب می‌شود. احتمال انتخاب خوشه‌ی j اُم برابر با وزن آن خوشه ($\phi^{(j)}$) است. شاخص(اندیس) خوشه‌ای که برای نمونه‌ی i اُم انتخاب شده، با $z^{(i)}$ نشان داده می‌شود.
- اگر نمونه‌ی i اُم به خوشه‌ی j اُم اختصاص داده شده باشد (یعنی $z^{(i)} = j$)، آنگاه موقعیت $\mathbf{x}^{(i)}$ این نمونه به‌صورت تصادفی از توزیع نرمال (گاوسی) با میانگین $\boldsymbol{\mu}^{(j)}$ و ماتریس کوواریانس $\boldsymbol{\Sigma}^{(j)}$ نمونه‌گیری می‌شود. این رابطه به صورت $\mathbf{x}^{(\mathbf{i})} \sim \mathcal{N}\left(\boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right)$ نوشته می‌شود.

خب، با چنین مدلی چه کارهایی می‌توان انجام داد؟ معمولاً وقتی داده‌های $\mathbf{X}$ را در اختیار دارید، اولین قدم این است که وزن‌ها ($\boldsymbol{\phi}$) و همه پارامترهای توزیع‌ها شامل میانگین‌ها $\boldsymbol{\mu}^{(1)}$ تا $\boldsymbol{\mu}^{(k)}$ و ماتریس‌های کوواریانس $\boldsymbol{\Sigma}^{(1)}$ تا $\boldsymbol{\Sigma}^{(k)}$ را تخمین بزنید. کلاس GaussianMixture در کتابخانه Scikit-Learn این کار را بسیار ساده می‌کند:

© 2025 Mohsen Zare. All rights reserved.

```
from sklearn.mixture import GaussianMixture

gm = GaussianMixture(n_components=3, n_init=10)
gm.fit(X)
```

بیایید نگاهی بیندازیم به پارامترهایی که الگوریتم تخمین زده است:

```
>>> gm.weights_
array([0.39025715, 0.40007391, 0.20966893])
>>> gm.means_
array([[ 0.05131611,  0.07521837],
       [-1.40763156,  1.42708225],
       [ 3.39893794,  1.05928897]])
>>> gm.covariances_
array([[[ 0.68799922,  0.79606357],
        [ 0.79606357,  1.21236106]],
       [[ 0.63479409,  0.72970799],
        [ 0.72970799,  1.1610351 ]],
       [[ 1.14833585, -0.03256179],
        [-0.03256179,  0.95490931]]])
```

عالی بود، همه‌چیز خوب پیش رفت! در واقع، دو مورد از سه خوشه با ۵۰۰ نمونه تولید شده بودند، در حالی که خوشه‌ی سوم تنها شامل ۲۵۰ نمونه است. بنابراین، وزن‌های واقعی خوشه‌ها به‌ترتیب ۰/۴، ۰/۴ و ۰/۲ هستند، و الگوریتم هم تقریباً همین مقادیر را پیدا کرده است. به‌طور مشابه، میانگین‌ها و ماتریس‌های کوواریانس واقعی نیز بسیار نزدیک به مقادیر به‌دست‌آمده توسط الگوریتم هستند. اما چطور؟ این کلاس از الگوریتم «انتظار-بیشینه‌سازی (*Expectation-Maximization* یا EM)» استفاده می‌کند، که شباهت‌های زیادی با الگوریتم k-means دارد. همچنین، این الگوریتم ابتدا پارامترهای خوشه‌ها را به‌صورت تصادفی مقداردهی اولیه می‌کند، سپس دو مرحله را تا زمان همگرایی تکرار می‌کند: ابتدا نمونه‌ها را به خوشه‌ها اختصاص می‌دهد (که به آن «مرحله‌ی انتظار (*expectation step*)» گفته می‌شود)، و سپس خوشه‌ها را به‌روزرسانی می‌کند (که به آن «مرحله‌ی بیشینه‌سازی (*maximization step*)» می‌گویند). آشنا به نظر می‌رسد، نه؟ در زمینه‌ی خوشه‌بندی، می‌توان الگوریتم EM را تعمیمی از k-means در نظر گرفت که نه‌تنها مرکز خوشه‌ها (از $\boldsymbol{\mu}^{(1)}$ تا $\boldsymbol{\mu}^{(k)}$) را پیدا می‌کند، بلکه اندازه، شکل و جهت‌گیری آن‌ها (از $\boldsymbol{\Sigma}^{(1)}$ تا $\boldsymbol{\Sigma}^{(k)}$) و همچنین وزن نسبی

© 2025 Mohsen Zare. All rights reserved.

خوشه‌ها (از $\phi^{(1)}$ تا $\phi^{(k)}$) را نیز تخمین می‌زند. اما برخلاف k-means، الگوریتم EM از «تخصیص نرم (*soft assignments*)» یا انتساب غیرقطعی استفاده می‌کند، نه «تخصیص سخت (*hard assignments*)» یا انتساب قطعی. در «مرحله‌ی انتظار»، برای هر نمونه، الگوریتم احتمال تعلق آن به هر خوشه را بر اساس پارامترهای فعلی خوشه‌ها تخمین می‌زند. سپس در «مرحله‌ی بیشینه‌سازی» ، هر خوشه با استفاده از «همهٔ» نمونه‌های موجود در مجموعه داده به‌روزرسانی می‌شود، به‌طوری‌که هر نمونه با توجه به احتمالی که برای تعلق به آن خوشه تخمین زده شده، وزن‌دهی می‌شود. این احتمالات را «مسئولیت (*responsibility*)» هر خوشه در قبال نمونه‌ها می‌نامند. در مرحله‌ی بیشینه‌سازی، به‌روزرسانی هر خوشه عمدتاً تحت تأثیر نمونه‌هایی قرار می‌گیرد که آن خوشه بیشترین مسئولیت را در قبال آنها دارد.

هُشدار

متأسفانه، درست مانند الگوریتم k-means، الگوریتم EM نیز ممکن است به راه‌حل‌های ضعیفی همگرا شود؛ بنابراین، لازم است چندین بار اجرا شود و فقط بهترین نتیجه حفظ شود. به همین دلیل، مقدار n_init را برابر با ۱۰ قرار دادیم. دقت کنید که به‌طور پیش‌فرض، مقدار n_init برابر با ۱ است.

می‌توانید بررسی کنید که آیا الگوریتم همگرا شده است یا نه و همچنین ببینید چند تکرار طول کشیده تا همگرا شود:

```
>>> gm.converged_
True
>>> gm.n_iter_
4
```

اکنون که تخمینی از مکان، اندازه، شکل، جهت‌گیری و وزنِ نسبی هر خوشه دارید، مدل می‌تواند به‌راحتی هر نمونه را به محتمل‌ترین خوشه اختصاص دهد(خوشه‌بندی سخت) یا احتمال تعلق آن به هر خوشه خاص را برآورد کند(خوشه‌بندی نرم). برای خوشه‌بندی سخت از متد predict() و برای خوشه‌بندی نرم از متد predict_proba() استفاده کنید:

© 2025 Mohsen Zare. All rights reserved.

```
>>> gm.predict(X)
array([0, 0, 1, ..., 2, 2, 2])
>>> gm.predict_proba(X).round(3)
array([[0.977, 0.   , 0.023],
       [0.983, 0.001, 0.016],
       [0.   , 1.   , 0.   ],
       ...,
       [0.   , 0.   , 1.   ],
       [0.   , 0.   , 1.   ],
       [0.   , 0.   , 1.   ]])
```

مدل مخلوطِ گاوسی یک «مدل مولد (*generative model*)» است، به این معنا که می‌توانید نمونه‌های جدیدی از آن تولید (یا نمونه‌گیری) کنید. توجه داشته باشید که این نمونه‌ها بر اساس شاخص خوشه مرتب شده‌اند:

```
>>> X_new, y_new = gm.sample(6)
>>> X_new
array([[-0.86944074, -0.32767626],
       [ 0.29836051,  0.28297011],
       [-2.8014927 , -0.09047309],
       [ 3.98203732,  1.49951491],
       [ 3.81677148,  0.53095244],
       [ 2.84104923, -0.73858639]])
>>> y_new
array([0, 0, 1, 2, 2, 2])
```

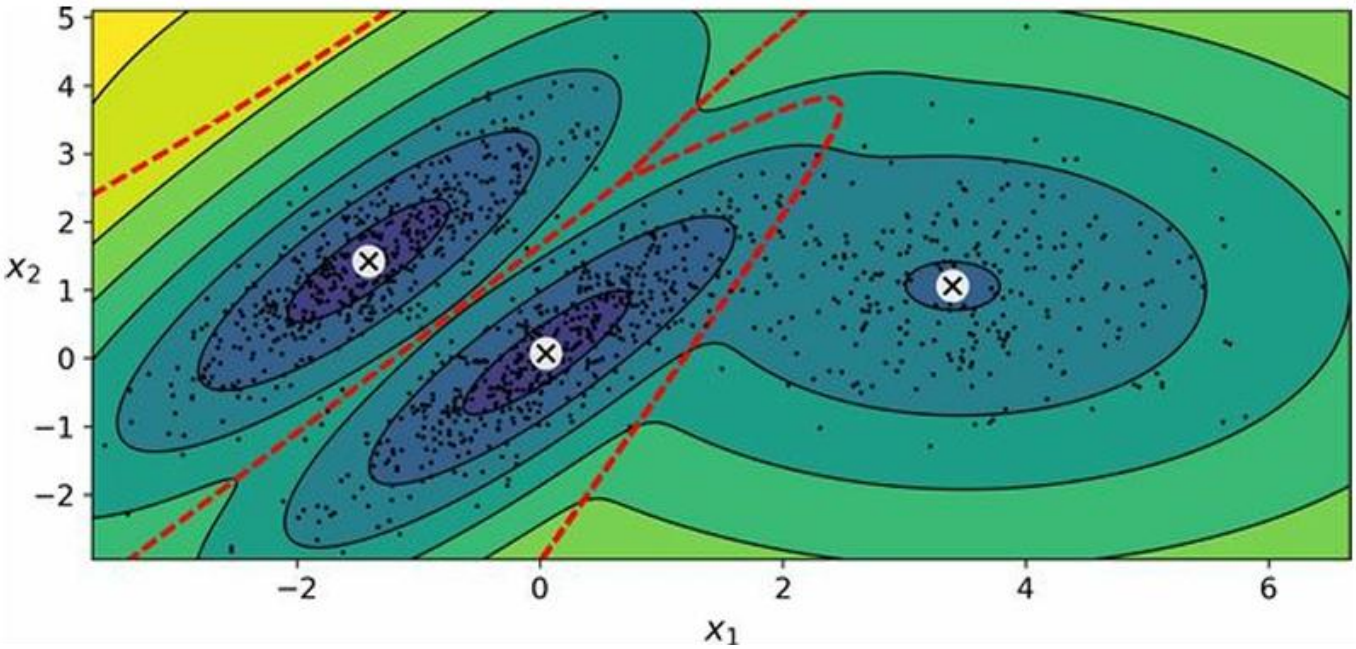
همچنین می‌توان چگالی مدل را در هر موقعیتی تخمین زد. این کار با استفاده از متد score_samples() انجام می‌شود: این متد برای هر نمونه‌ای که به آن داده می‌شود، لگاریتم «تابع چگالی احتمال (PDF: *probability density function*)» را در آن نقطه تخمین می‌زند. هرچه مقدار این امتیاز بیشتر باشد، چگالی در آن نقطه بالاتر است:

```
>>> gm.score_samples(X).round(2)
array([-2.61, -3.57, -3.33, ..., -3.51, -4.4 , -3.81])
```

© 2025 Mohsen Zare. All rights reserved.

اگر نمودارِ نمایی (exponential) این امتیازها را محاسبه کنید، مقدار تابع چگالی احتمال (PDF) را در محل نمونه‌های داده‌شده به دست می‌آورید. این مقادیر «احتمال» نیستند، بلکه «چگالی احتمال» هستند: بنابراین می‌توانند هر مقدار مثبتی داشته باشند، نه فقط مقداری بین ۰ و ۱. برای تخمینِ احتمالِ اینکه یک نمونه در ناحیه‌ای خاص قرار بگیرد، باید PDF را روی آن ناحیه انتگرال بگیرید. (اگر PDF را روی کل فضایِ ممکنِ نمونه‌ها انتگرال بگیرید، نتیجه برابر ۱ خواهد بود.)

شکل ۹-۱۶ میانگین خوشه‌ها، مرزهای تصمیم (خط‌چین‌ها) و مرز منحنی‌های چگالی این مدل را نشان می‌دهد.



شکل ۹-۱۶. میانگین‌های خوشه‌ای، مرزهای تصمیم‌گیری و خطوط چگالی یک مدل مخلوط گاوسی آموزش‌دیده

عالی! الگوریتم به وضوح یک راه حل عالی پیدا کرد. البته، ما با تولید داده‌ها با استفاده از مجموعه‌ای از توزیع‌های گاوسی دوبعدی، کار آن را آسان کردیم (متأسفانه، داده‌های دنیای واقعی همیشه آنقدر گاوسی و کم‌بعد نیستند). ما همچنین تعداد صحیح خوشه‌ها را به الگوریتم دادیم. وقتی ابعاد زیادی، یا خوشه‌های زیادی، یا نمونه‌های کمی وجود داشته باشد، EM می‌تواند برای همگرایی به راه حل بهینه تلاش کند. ممکن است لازم باشد با محدود کردن تعداد پارامترهایی که الگوریتم باید یاد بگیرد، سختی کار را کاهش دهید. یک راه برای انجام این کار، محدود کردن دامنه شکل‌ها و جهت‌هایی است که خوشه‌ها می‌توانند داشته باشند. این کار را می‌توان با اعمال محدودیت بر ماتریس‌های کوواریانس انجام داد. برای انجام این کار، هایپرپارامتر covariance_type را روی یکی از مقادیر زیر تنظیم کنید:

© 2025 Mohsen Zare. All rights reserved.

"spherical"

همه خوشه‌ها باید کروی باشند، اما می‌توانند قطرهای متفاوتی داشته باشند (یعنی واریانس‌های متفاوتی داشته باشند).

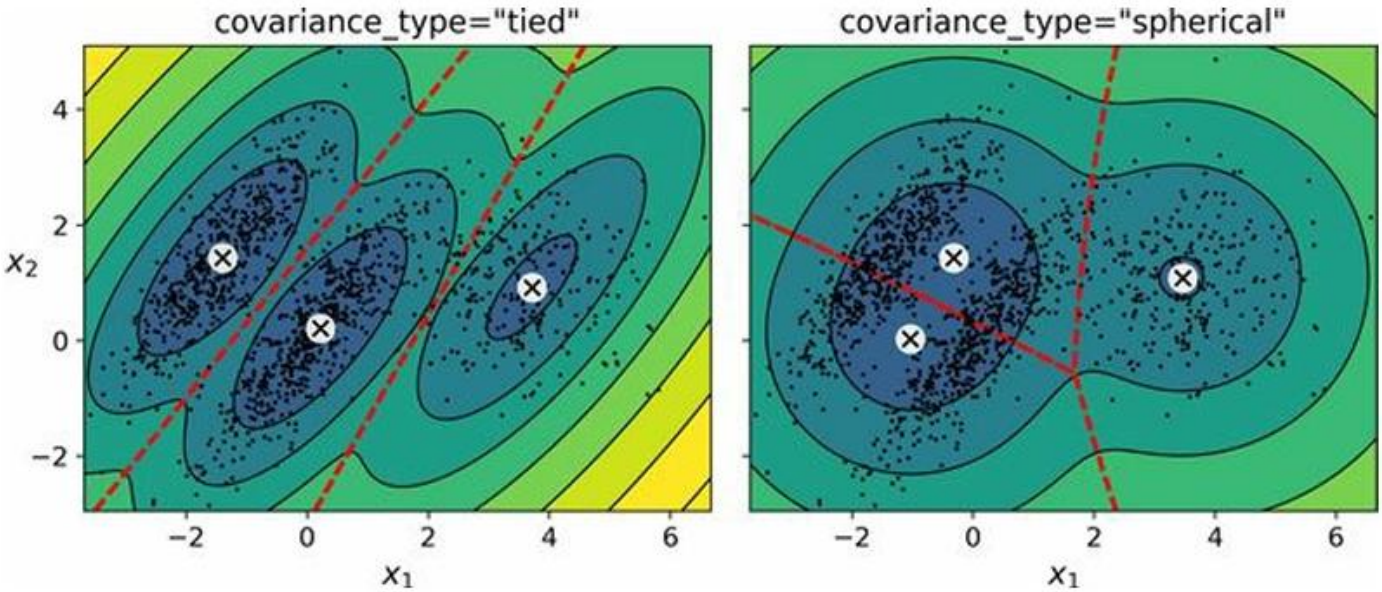
"diag"

خوشه‌ها می‌توانند هر شکل بیضوی با هر اندازه‌ای را به خود بگیرند، اما محورهای بیضوی باید موازی با محورهای مختصات باشند (یعنی ماتریس‌های کوواریانس باید قطری باشند).

"tied"

همه خوشه‌ها باید شکل، اندازه و جهت‌گیری بیضوی یکسانی داشته باشند (یعنی همه خوشه‌ها ماتریس کوواریانس یکسانی داشته باشند).

به طور پیش‌فرض، covariance_type برابر با "full" است، به این معنی که هر خوشه می‌تواند هر شکل، اندازه و جهتی را به خود بگیرد (ماتریس کوواریانس نامحدود خود را دارد). شکل ۹-۱۷ راه‌حل‌های یافت شده توسط الگوریتم EM را هنگامی که covariance_type روی "tied" یا "spherical" تنظیم شده است، رسم می‌کند.



شکل ۹-۱۷. مخلوط‌های گاوسی برای خوشه‌های گره خورده (چپ) و خوشه‌های کروی (راست)

© 2025 Mohsen Zare. All rights reserved.

نکته

پیچیدگی محاسباتی آموزش یک مدل GaussianMixture به تعداد نمونه‌ها m، تعداد ابعاد n، تعداد خوشه‌ها k و محدودیت‌های ماتریس‌های کوواریانس بستگی دارد. اگر covariance_type ، "spherical" یا "diag" باشد، با فرض اینکه داده‌ها ساختار خوشه‌بندی دارند، برابر با $O(kmn)$ است. اگر covariance_type ، "tied" یا "full" باشد، برابر با $O(kmn^2 + kn^3)$ است، بنابراین برای تعداد زیادی از ویژگی‌ها مقیاس‌پذیر نخواهد بود.

مدل‌های مخلوط گاوسی همچنین می‌توانند برای تشخیص ناهنجاری استفاده شوند. چگونگی آن را در بخش بعدی خواهیم دید.

© 2025 Mohsen Zare. All rights reserved.

استفاده از مخلوط‌های گاوسی برای تشخیص ناهنجاری

استفاده از مدل مخلوط گاوسی برای تشخیص ناهنجاری بسیار ساده است: هر نمونه‌ای که در یک ناحیه با چگالی کم قرار دارد، می‌تواند یک ناهنجاری در نظر گرفته شود. شما باید آستانه چگالی مورد نظر خود را تعریف کنید. به عنوان مثال، در یک شرکت تولیدی که سعی در تشخیص محصولات معیوب دارد، نسبت محصولات معیوب معمولاً مشخص است. فرض کنید برابر با ۲ % است. سپس آستانه چگالی را مقداری تعیین می‌کنید که منجر به قرار گرفتن ۲ % از نمونه‌ها در نواحی زیر آن چگالی آستانه می‌شود. اگر متوجه شدید که موارد مثبت کاذب زیادی دارید (یعنی محصولات کاملاً خوبی که به عنوان معیوب علامت‌گذاری شده‌اند)، می‌توانید آستانه را کاهش دهید. برعکس، اگر موارد منفی کاذب زیادی دارید (یعنی محصولات معیوبی که سیستم آنها را به عنوان معیوب علامت‌گذاری نمی‌کند)، می‌توانید آستانه را افزایش دهید. این بده بستان معمول دقت/فراخوانی است (به فصل ۳ مراجعه کنید). در اینجا نحوه شناسایی داده‌های پرت با استفاده از کمترین چگالی صدک چهارم به عنوان آستانه (یعنی تقریباً ۴ % از موارد به عنوان ناهنجاری علامت‌گذاری می‌شوند) آورده شده است:

```
densities = gm.score_samples(X)
density_threshold = np.percentile(densities, 2)
anomalies = X[densities < density_threshold]
```

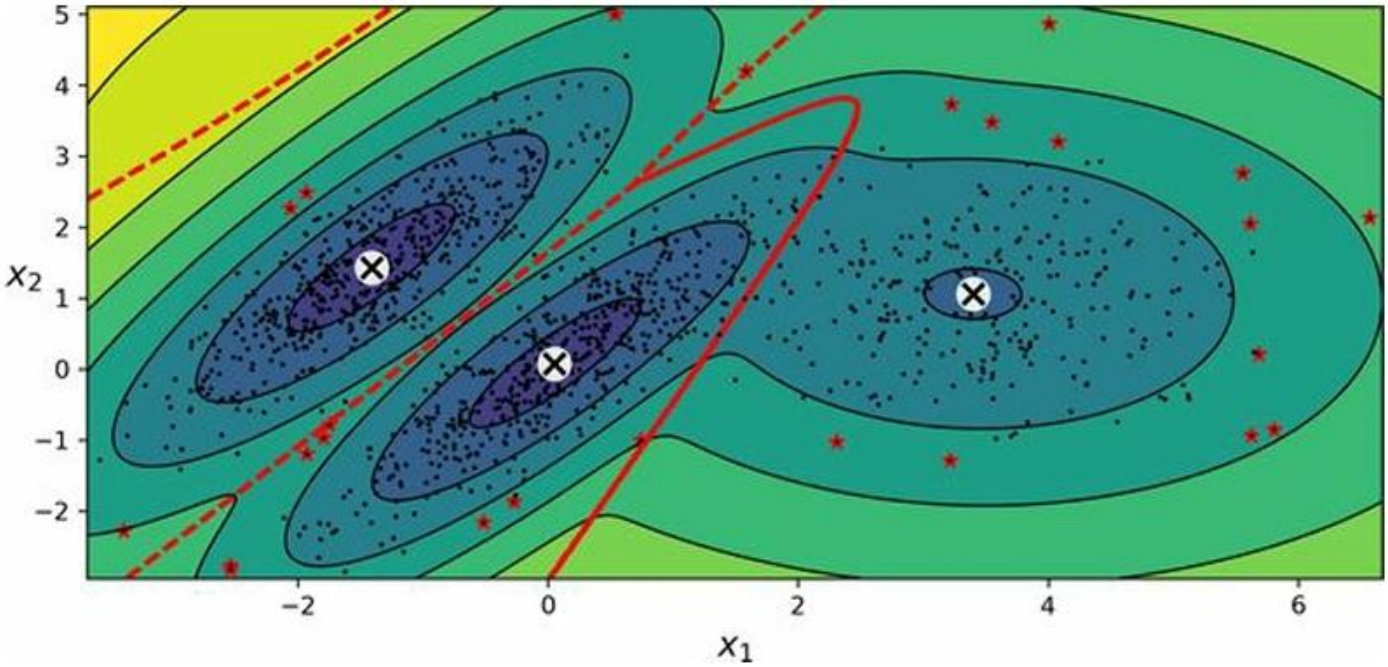
شکل ۹-۱۸ این ناهنجاری‌ها را به صورت ستاره نشان می‌دهد.

یک مسئله‌ی نزدیک به آن، «تشخیص موارد جدید (*novelty detection*)» است: این کار با تشخیص ناهنجاری متفاوت است، زیرا فرض می‌شود الگوریتم روی یک مجموعه داده «تمیز» آموزش دیده است که توسط داده‌های پرت آلوده نشده است، در حالی که تشخیص ناهنجاری این فرض را در نظر نمی‌گیرد. در واقع، تشخیص داده‌های پرت اغلب برای پاکسازی یک مجموعه داده استفاده می‌شود.

© 2025 Mohsen Zare. All rights reserved.

نکته

مدل‌های مخلوط گاوسی سعی می‌کنند همه داده‌ها، از جمله داده‌های پرت را برازش دهند؛ اگر تعداد زیادی از آنها را داشته باشید، این امر باعث می‌شود که دیدگاه مدل نسبت به «نرمال بودن» دچار بایاس شود و برخی از داده‌های پرت ممکن است به اشتباه نرمال در نظر گرفته شوند. اگر این اتفاق بیفتد، می‌توانید یک بار سعی کنید مدل را برازش دهید، از آن برای شناسایی و حذف شدیدترین داده‌های پرت استفاده کنید، سپس دوباره مدل را روی مجموعه داده پاک‌سازی شده برازش دهید. رویکرد دیگر استفاده از روش‌های تخمین کوواریانسِ robust است (به کلاس EllipticEnvelope مراجعه کنید).



شکل ۹-۱۸. تشخیص ناهنجاری با استفاده از مدل مخلوط گاوسی

درست مانند الگوریتم k-means، الگوریتم GaussianMixture نیز نیاز دارد که تعداد خوشه‌ها را مشخص کنید. پس چگونه می‌توان این تعداد را تعیین کرد؟

© 2025 Mohsen Zare. All rights reserved.

انتخاب تعداد خوشه‌ها

در الگوریتم k-means، می‌توان از معیارهایی مانند اینرسی یا امتیاز سیلوئت برای انتخاب تعداد مناسب خوشه‌ها استفاده کرد. اما در مدل‌های اختلاط گاوسی، استفاده از این معیارها ممکن نیست، زیرا وقتی خوشه‌ها کروی نباشند یا اندازه‌های متفاوتی داشته باشند، این معیارها قابل‌اعتماد نیستند. در عوض، می‌توانید مدلی را انتخاب کنید که یکی از «معیار اطلاعات نظری (*Theoretical Information Criterion* (TIC))» مانند «معیارِ اطلاعات مبتنی بِیز یا بِیزیَن (*Bayesian Information Criterion* (BIC))» و یا «معیار اطلاعات آکائیکه (Akaike Information Criterion (AIC))» را که در معادله ۹-۱ تعریف شده‌اند، به حداقل می‌رساند.

معادله ۹-۱. معیار اطلاعات بیزی (BIC) و معیار اطلاعات آکائیک (AIC)

$$\mathrm{BIC} = \log(m)\,.\,p - 2\log(\hat{L})$$

$$\mathrm{AIC} = 2\,p - 2\log(\hat{L})$$

در این معادلات:

- m تعداد نمونه‌ها (instances) است، همان‌طور که همیشه بوده.
- p تعداد پارامترهایی است که مدل آن‌ها را یاد گرفته است.
- $\hat{L}$ مقدار ماکسیمم شده‌ی «تابع احتمال (*likelihood function*)» مدل است.

© 2025 Mohsen Zare. All rights reserved.

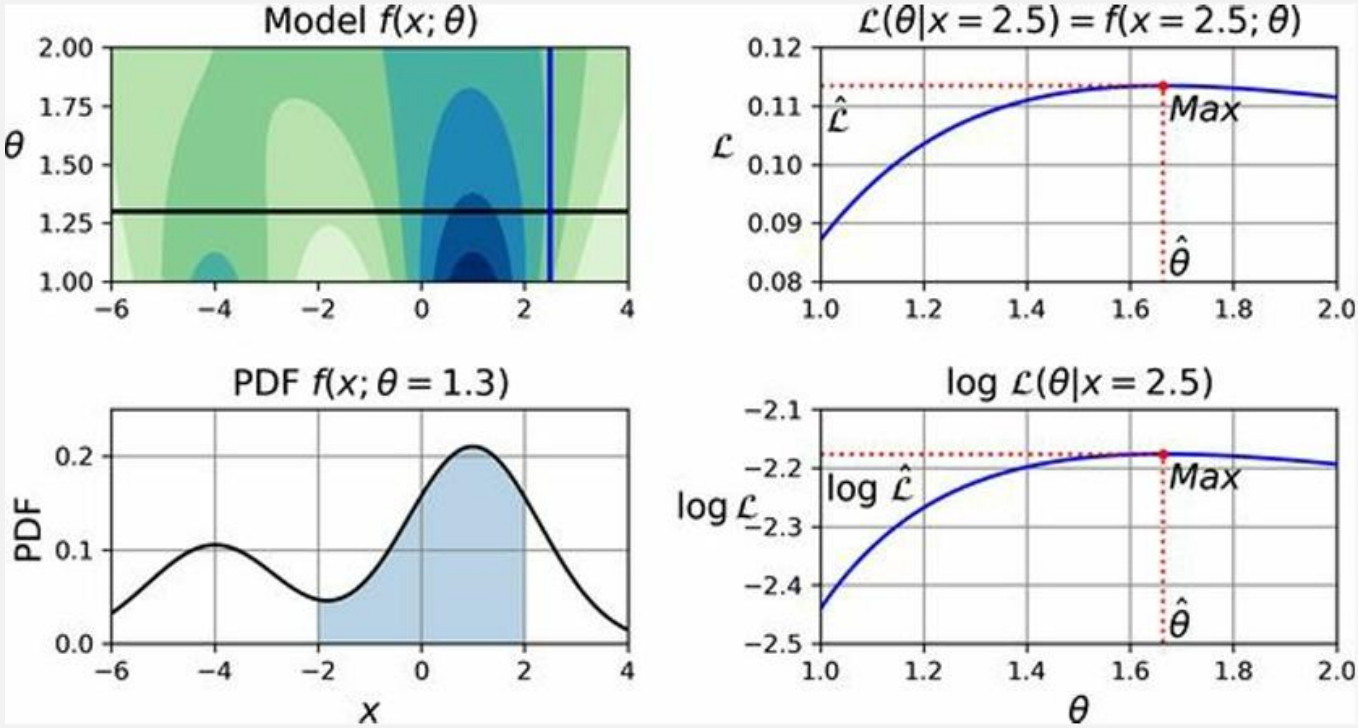
هم معیار BIC و هم AIC مدل‌هایی را که پارامترهای بیشتری برای یادگیری دارند (مثلاً تعداد خوشه‌های بیشتر) جریمه می‌کنند و مدل‌هایی را که به‌خوبی با داده‌ها منطبق(برازش) می‌شوند، پاداش می‌دهند. این دو معیار اغلب یک مدل مشابه را انتخاب می‌کنند. اما زمانی که انتخاب آن‌ها متفاوت باشد، معمولاً مدلی که توسط BIC انتخاب می‌شود ساده‌تر است (یعنی پارامترهای کمتری دارد) نسبت به مدلی که AIC انتخاب می‌کند، اما معمولاً به‌خوبی AIC داده‌ها را برازش نمی‌دهد—به‌ویژه در مجموعه داده‌های بزرگ.

تابع احتمال

اصطلاحات «*probability*» و «*likelihood*» هردو به معنی «احتمال» هستند و در زبان روزمره اغلب به‌جای یکدیگر استفاده می‌شوند، اما در آمار معناهای کاملاً متفاوتی دارند. وقتی یک مدل آماری با پارامترهایی مانند $\boldsymbol{\theta}$ داشته باشیم، واژه‌ی «*probability*» برای توصیف میزان محتمل بودن یک نتیجه‌ی آینده‌ی $\mathbf{x}$ (با دانستن مقادیر پارامتر $\boldsymbol{\theta}$) به کار می‌رود. در مقابل، واژه‌ی «*likelihood*» برای توصیف میزان محتمل بودن یک مجموعه خاص از مقادیر پارامتر $\boldsymbol{\theta}$ پس از آنکه نتیجه‌ی $\mathbf{x}$ معلوم شده است، استفاده می‌شود. یک مدل ترکیبی یک‌بعدی را در نظر بگیرید که از دو توزیع گاوسی تشکیل شده است، یکی با میانگین ۴ و دیگری با میانگین ۱+. برای سادگی، این مدل آزمایشی تنها یک پارامتر θ دارد که انحراف معیار هر دو توزیع را کنترل می‌کند. نمودار کانتور بالا-چپ در شکل ۹-۱۹، کل مدل $f(x;\theta)$ را به‌صورت تابعی از x و θ نمایش می‌دهد. برای تخمینِ توزیعِ احتمالی یک نتیجه‌ی آینده x، باید پارامتر مدل θ را تعیین کنید. به‌عنوان‌مثال، اگر مقدار θ را برابر با ۱٫۳ قرار دهید (خط افقی)، تابع چگالی احتمال $f(x;\theta = 1.3)$ را به دست می‌آورید که در نمودار پایین-چپ نمایش داده شده است. فرض کنید می‌خواهید احتمال این را تخمین بزنید که مقدار x بین ۲- و ۲+ قرار گیرد. برای این کار باید انتگرال تابع چگالی احتمال (PDF) را در این بازه محاسبه کنید (یعنی مساحت ناحیه سایه‌خورده). اما اگر مقدار θ را ندانید و در عوض فقط یک مشاهده داشته باشید، مثلاً $x = 2.5$ (که با خط عمودی در نمودار بالا-چپ نشان داده شده)، چه باید کرد؟ در این حالت، به جای داشتن تابع احتمال، تابع احتمال $\mathcal{L}(\theta|x = 2.5) = f(x = 2.5;\ \theta)$ را به‌دست می‌آورید، که در نمودار بالا-راست نشان داده

© 2025 Mohsen Zare. All rights reserved.

شده است. این تابع نشان می‌دهد که برای مقدار مشاهده‌شده‌ی $x = 2.5$، کدام مقادیر θ محتمل‌تر هستند.



شکل ۹-۱۹. تابع پارامتریک یک مدل (بالا-چپ)، و چند تابع مشتق‌شده: یک تابع چگالی احتمال (پایین-چپ)، یک تابع احتمال (بالا-راست)، و یک تابع لگاریتم احتمال (پایین-راست)

به‌طور خلاصه، تابع چگالی احتمال (PDF) تابعی از x است (با θ ثابت)، در حالی‌که تابع احتمال، تابعی از θ است (با x ثابت). درک این نکته مهم است که تابع احتمال یک توزیع احتمالی «نیست»: اگر تابع توزیع احتمال را روی تمام مقادیر ممکن x انتگرال بگیرید، نتیجه همیشه برابر ۱ خواهد بود، اما اگر تابع احتمال را روی تمام مقادیر ممکن θ انتگرال بگیرید، نتیجه می‌تواند هر مقدار مثبتی باشد. با داشتن یک مجموعه‌داده $\mathbf{X}$، یکی از مسئله‌های رایج این است که محتمل‌ترین مقادیر برای پارامترهای مدل را تخمین بزنیم. برای انجام این کار، باید مقادیری را پیدا کنید که تابع احتمال را با توجه به $\mathbf{X}$ بیشینه می‌کنند. در این مثال، اگر تنها یک نمونه $x = 2.5$ را مشاهده کرده باشید، «تخمینِ ماکسیمُم احتمال (*Maximum Likelihood Estimate* (MLE))» برای θ برابر است با $\hat{\theta} = 1.5$. اگر یک توزیع احتمال پیشین g برای θ وجود داشته باشد، می‌توان آن را در نظر گرفت و به‌جای بیشینه‌سازی صِرفِ تابعِ احتمال $\mathcal{L}(\theta|x)$، عبارت $\mathcal{L}(\theta|x)g(\theta)$ را بیشینه کرد. این روش «تخمینِ بیشینه‌ی پَسین (*Maximum a-Posteriori* یا MAP)» نام دارد. از آنجا

© 2025 Mohsen Zare. All rights reserved.

که MAP مقادیر پارامترها را محدود می‌کند، می‌توان آن را نسخه‌ای منظم‌سازی‌شده (regularized) از MLE در نظر گرفت.

توجه داشته باشید که بیشینه‌سازی تابع احتمال معادل با بیشینه‌سازی لگاریتم آن است (که در نمودار پایین سمت راست در شکل ۹-۱۹ نشان داده شده است). در واقع، لگاریتم یک تابع صعودیِ یکنوا(اکیداً صعودی) است، بنابراین اگر θ لگاریتمِ احتمال را بیشینه کند، خود تابع احتمال را نیز بیشینه خواهد کرد. در عمل، معمولاً بیشینه‌سازی لگاریتم احتمال آسان‌تر است. برای مثال، اگر چند مشاهده مستقل از نمونه‌های $x^{(1)}$ تا $x^{(m)}$ داشته باشید، لازم است مقداری از θ را پیدا کنید که حاصل‌ضرب تابع‌های تحقق‌پذیریِ مربوط به هر مشاهده را بیشینه کند. اما به لطف ویژگی جادویی لگاریتم که حاصل‌ضرب‌ها را به جمع تبدیل می‌کند ($log(ab) = log(a) + log(b)$)، می‌توان به‌جای آن، جمع لگاریتم تابع‌های تحقق‌پذیری را بیشینه کرد که بسیار ساده‌تر است.

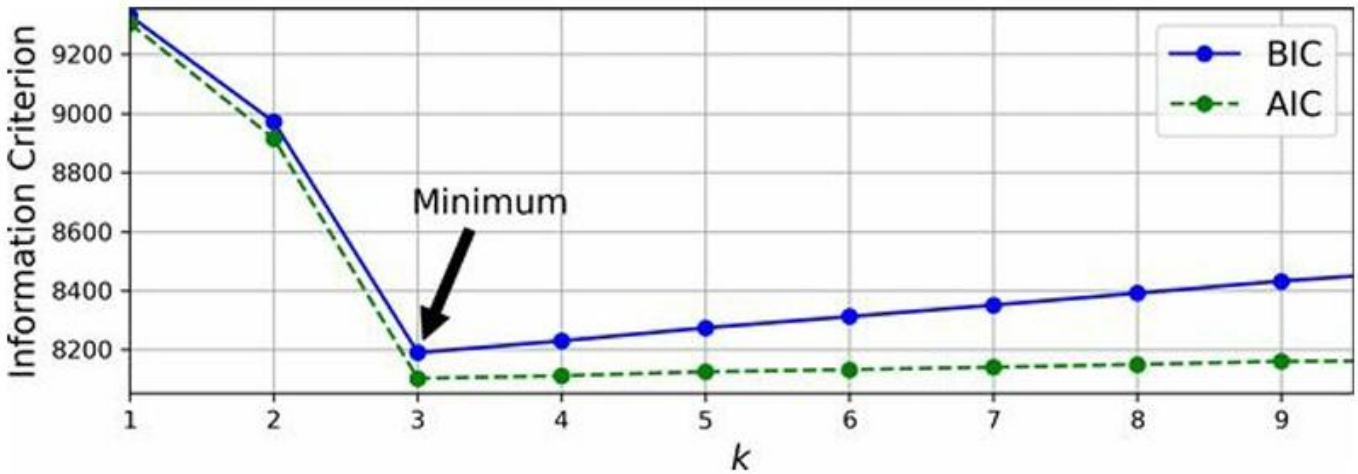
وقتی مقدار $\hat{\theta}$ را تخمین زدید—یعنی همان مقداری از θ که تابع احتمال را بیشینه می‌کند—آماده‌اید تا مقدار $\hat{L} = L(\hat{\theta}, \mathrm{X})$ را محاسبه کنید. این مقدار برای محاسبه معیارهای AIC و BIC استفاده می‌شود و می‌توانید آن را به‌عنوان معیاری در نظر بگیرید که نشان می‌دهد مدل تا چه حد داده‌ها را به‌خوبی توصیف می‌کند(تا چه میزان با داده‌ها برازش دارد).

برای محاسبه معیارهای BIC و AIC، می‌توانید از متدهای bic() و aic() استفاده کنید:

```
>>> gm.bic(X)
8189.747000497186
>>> gm.aic(X)
8102.521720382148
```

شکل ۹-۲۰ مقدار BIC را برای تعداد مختلف خوشه‌ها (k) نشان می‌دهد. همان‌طور که مشاهده می‌کنید، هر دو معیار BIC و AIC زمانی که $k = 3$ است به کمترین مقدار خود می‌رسند؛ بنابراین، به‌احتمال زیاد این مقدار بهترین انتخاب است.

© 2025 Mohsen Zare. All rights reserved.



شکل ۹-۲۰. مقادیر AIC و BIC برای تعدادهای مختلف خوشه‌ها (k)

© 2025 Mohsen Zare. All rights reserved.

مدل‌های مخلوطِ گاوسیِ بیزی

به‌جای جستجوی دستی برای یافتن تعداد بهینه خوشه‌ها، می‌توانید از کلاس BayesianGaussianMixture استفاده کنید. این کلاس قادر است به خوشه‌های غیرضروری وزن‌هایی برابر (یا نزدیک) به صفر اختصاص دهد. تعداد خوشه‌ها (n_components) را روی مقداری تنظیم کنید که به‌دلایل منطقی فکر می‌کنید بیشتر از تعداد خوشه‌های بهینه است (این کار فرض می‌کند که شما حداقل آشنایی نسبی با مسئله دارید)، و الگوریتم به‌طور خودکار خوشه‌های غیرضروری را حذف خواهد کرد. برای مثال، بیایید تعداد خوشه‌ها را روی ۱۰ تنظیم کنیم و ببینیم چه اتفاقی می‌افتد:

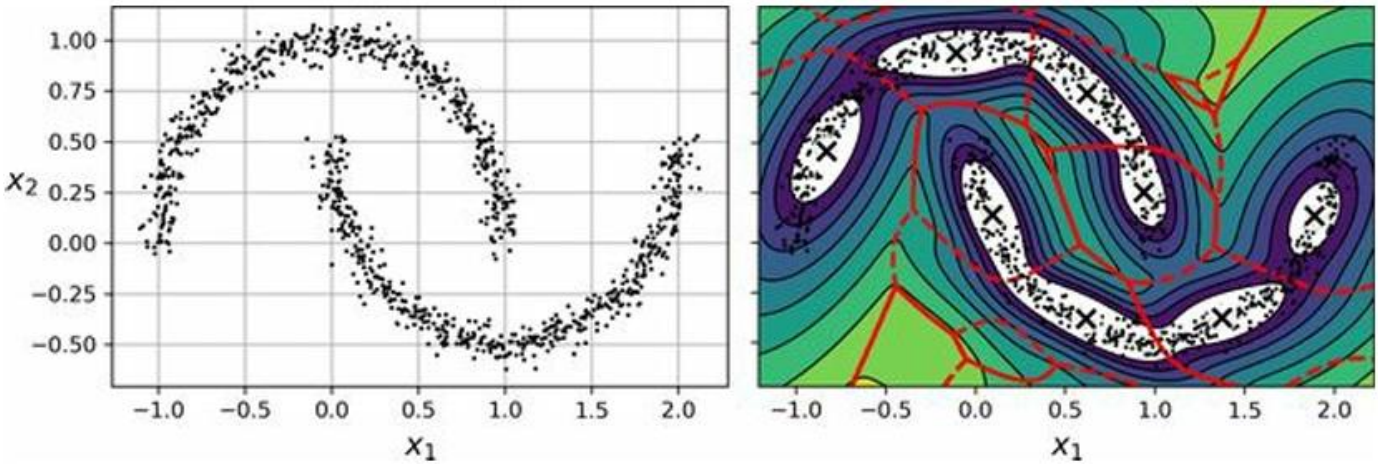
```
>>> from sklearn.mixture import BayesianGaussianMixture
>>> bgm = BayesianGaussianMixture(n_components=10, n_init=10, random_state=42)
>>> bgm.fit(X)
>>> bgm.weights_.round(2)
array([0.4 , 0.21, 0.4 , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ])
```

عالی: الگوریتم به‌صورت خودکار تشخیص داد که تنها سه خوشه لازم است و خوشه‌های به‌دست‌آمده تقریباً مشابه همان‌هایی هستند که در شکل ۹-۱۶ مشاهده شد.

یک نکته نهایی درباره مدل‌های مخلوط گاوسی: اگرچه این مدل‌ها روی خوشه‌هایی با شکل‌های بیضوی عملکرد بسیار خوبی دارند، اما وقتی خوشه‌ها شکل‌های بسیار متفاوتی داشته باشند، عملکرد خوبی از خود نشان نمی‌دهند. برای مثال، بیایید ببینیم وقتی از مدل مخلوطِ گاوسی بیزی برای خوشه‌بندی مجموعه داده «moons» استفاده کنیم، چه اتفاقی می‌افتد (نگاه کنید به شکل ۹-۲۱).

اوه! الگوریتم به شدت به دنبال خوشه‌های بیضوی بود، بنابراین به جای دو خوشه، هشت خوشه مختلف پیدا کرد. تخمین چگالی چندان بد نیست، بنابراین ممکن است این مدل برای تشخیص ناهنجاری‌ها قابل استفاده باشد، اما در شناسایی دو مورد «ماه» موفق نبود. برای پایان این فصل، بیایید نگاهی سریع به چند الگوریتمی بیندازیم که قادر به کار با خوشه‌هایی با شکل دلخواه هستند.

© 2025 Mohsen Zare. All rights reserved.



شکل ۹-۲۱. برازش مدل مخلوط گاوسی به خوشه‌های غیر بیضوی شکل

© 2025 Mohsen Zare. All rights reserved.

سایر الگوریتم‌ها برای تشخیص «ناهنجاری» و «موارد جدید»

Scikit-Learn الگوریتم‌های دیگری را برای تشخیص ناهنجاری یا نوآوری پیاده‌سازی کرده است:

حداقل تعیین‌کننده‌ی کوواریانس (*minimum covariance determinant* یا *Fast-MCD*)

این الگوریتم که توسط کلاس EllipticEnvelope پیاده‌سازی شده، برای شناسایی داده‌های پرت بسیار مفید است، به‌خصوص برای پاک‌سازی داده‌ها. فرض می‌کند نمونه‌های نرمال (داده‌های درونِ‌محدوده یا inliers) از یک توزیع گاوسی منفرد (نه مخلوط) تولید شده‌اند. همچنین فرض می‌کند مجموعه داده شامل داده‌های پرت هستند که از این توزیعِ گاوسی تولید نشده‌اند. وقتی الگوریتم پارامترهای توزیع گاوسی (یعنی شکل پوشش بیضوی اطراف داده‌های درونِ‌محدوده) را تخمین می‌زند، با دقت نمونه‌هایی را که احتمالاً داده پرت هستند نادیده می‌گیرد. این تکنیک برآورد بهتری از پوشش بیضوی ارائه می‌دهد و در نتیجه الگوریتم را در شناسایی داده‌های پرت قوی‌تر می‌کند.

جنگل ایزوله (*Isolation forest*)

این الگوریتمی کارآمد برای شناسایی نقاط پرت، به ویژه در داده‌های با ابعاد بالا است. الگوریتم یک جنگل تصادفی می‌سازد که در آن هر درخت تصمیم به‌صورت تصادفی رشد می‌کند: در هر گره، یک ویژگی به‌صورت تصادفی انتخاب می‌شود، سپس یک مقدار آستانهٔ تصادفی (بین حداقل و حداکثر مقدار آن ویژگی) برای تقسیم داده‌ها به دو بخش انتخاب می‌شود. داده‌ها به تدریج به قطعات کوچک‌تری تقسیم می‌شوند تا جایی که همه نمونه‌ها از بقیه جدا و منفرد می‌شوند. نقاط پرت یا ناهنجاری‌ها معمولاً از سایر نمونه‌ها فاصله زیادی دارند، بنابراین به طور متوسط (در تمام درختان تصمیم) آن‌ها در مراحل کمتری نسبت به نمونه‌های عادی جدا می‌شوند.

© 2025 Mohsen Zare. All rights reserved.

فاکتور داده‌های پرت محلی (*Local outlier factor* یا *LOF*)

این الگوریتم نیز برای شناسایی نقاط پرت مناسب است. این روش چگالی نمونه‌ها در اطراف یک نمونه خاص را با چگالی اطراف همسایگان آن مقایسه می‌کند. یک ناهنجاری معمولاً از k-نزدیک‌ترین همسایه خودش منزوی‌تر است.

SVM تک کلاسه (*One-class SVM*)

این الگوریتم برای تشخیص موارد جدید (novelty detection) مناسب‌تر است. به خاطر بیاورید که یک طبقه‌بند SVM کرنل‌دار(هسته‌دار)، ابتدا همه نمونه‌ها را به‌طور ضمنی به یک فضای با ابعاد بالا نگاشت (mapping) می‌کند و سپس دو کلاس را در این فضای با استفاده از یک طبقه‌بند خطی SVM جدا می‌کند (رجوع کنید به فصل ۵). از آنجایی که در اینجا تنها یک کلاس نمونه داریم، الگوریتم «SVM تک-کلاسه» تلاش می‌کند نمونه‌ها را در فضای با ابعاد بالا از مبدا (origin) جدا کند. در فضای اصلی، این کار معادل یافتن یک ناحیه کوچک است که تمام نمونه‌ها را در بر می‌گیرد. اگر نمونه جدیدی در این ناحیه قرار نگیرد، آن نمونه به عنوان ناهنجاری شناخته می‌شود. چندین هایپرپارامتر برای تنظیم وجود دارد: پارامترهای معمول برای یک SVM کرنل‌دار، به‌علاوه یک هایپرپارامتر حاشیه که نشان‌دهنده احتمال این است که نمونه جدید به اشتباه به عنوان نمونه جدید(novel) در نظر گرفته شود، در حالی که در واقع نمونه‌ای عادی است. این الگوریتم عملکرد بسیار خوبی دارد، به ویژه با داده‌های با ابعاد بالا، اما مانند همه SVMها، برای مجموعه داده‌های بسیار بزرگ مقیاس‌پذیر نیست.

PCA و سایر تکنیک‌های کاهش ابعاد با متد *inverse_transform()*

اگر خطای بازسازی یک نمونه عادی را با خطای بازسازی یک ناهنجاری مقایسه کنید، معمولاً خطای بازسازی ناهنجاری بسیار بزرگ‌تر خواهد بود. این یک روش ساده و اغلب بسیار کارآمد برای شناسایی ناهنجاری است (برای نمونه‌ای از این روش به تمرین‌های این فصل مراجعه کنید).

© 2025 Mohsen Zare. All rights reserved.

تمرین‌ها

۱. خوشه‌بندی را چگونه تعریف می‌کنید؟ چند نمونه از الگوریتم‌های خوشه‌بندی را نام ببرید.

۲. مهم‌ترین کاربردهای الگوریتم‌های خوشه‌بندی کدام‌اند؟

۳. هنگام استفاده از k-means، دو روش برای انتخاب تعداد مناسب خوشه‌ها را توضیح دهید.

۴. انتشار برچسب چیست؟ به چه دلیل و با چه روشی پیاده‌سازی می‌شود؟

۵. دو الگوریتم خوشه‌بندی که قابلیت مقیاس‌پذیری روی مجموعه‌داده‌های بزرگ دارند را نام ببرید. همچنین دو الگوریتم که نواحی با چگالی بالا را شناسایی می‌کنند معرفی کنید.

۶. مثالی از موقعیتی بیاورید که در آن یادگیری فعال سودمند باشد. چگونه می‌توان آن را اجرا کرد؟

۷. چه تفاوتی میان تشخیص ناهنجاری و شناسایی داده‌های جدید وجود دارد؟

۸. مدل مخلوط گاوسی چیست و برای چه کاربردهایی به کار می‌رود؟

۹. دو روش برای تعیین تعداد خوشه‌های مناسب در مدل مخلوط گاوسی را معرفی کنید.

۱۰. مجموعه‌داده‌ی کلاسیک Olivetti Faces شامل ۴۰۰ تصویر خاکستری با اندازه ۶۴×۶۴ پیکسل از چهره‌ها است. هر تصویر به یک بردار تک‌بعدی با طول ۴,۰۹۶ تبدیل شده است. این تصاویر مربوط به ۴۰ فرد مختلف هستند که از هر فرد ۱۰ عکس گرفته شده است. مسئله‌ی رایج در این مجموعه‌داده، آموزش مدلی است که بتواند پیش‌بینی کند هر تصویر مربوط به کدام فرد است. ابتدا مجموعه‌داده را با استفاده از تابع sklearn.datasets.fetch_olivetti_faces() بارگذاری کنید و آن را به سه بخش آموزشی، اعتبارسنجی و آزمایشی تقسیم نمایید (توجه داشته باشید که داده‌ها از قبل در بازه‌ی ۰ تا ۱ نرمال‌سازی شده‌اند). با توجه به کوچک بودن این مجموعه‌داده، بهتر است از نمونه‌گیری طبقه‌بندی‌شده استفاده کنید تا تعداد تصاویر هر فرد در هر بخش برابر باشد. سپس با استفاده از k-means تصاویر را خوشه‌بندی کنید و با بهره‌گیری از یکی از روش‌های معرفی‌شده در این فصل، تعداد مناسب خوشه‌ها را انتخاب کنید. در نهایت خوشه‌ها را مصورسازی کنید: آیا در هر خوشه چهره‌های مشابه یکدیگر قرار گرفته‌اند؟

© 2025 Mohsen Zare. All rights reserved.

۱۱. با استفاده از مجموعه‌داده‌ی چهره‌های Olivetti، ابتدا یک طبقه‌بند آموزش دهید تا بتواند پیش‌بینی کند هر تصویر متعلق به کدام فرد است و عملکرد آن را روی مجموعه‌ی اعتبارسنجی بسنجید. در مرحله‌ی بعد، از k-means به‌عنوان روشی برای کاهش ابعاد استفاده کنید و یک طبقه‌بند را روی داده‌های کاهش‌یافته آموزش دهید. سپس تعداد خوشه‌هایی را جستجو کنید که بهترین عملکرد را برای طبقه‌بند فراهم می‌کنند. ببینید با این رویکرد به چه سطحی از دقت دست پیدا می‌کنید. در نهایت، بررسی کنید اگر ویژگی‌های کاهش‌یافته‌ی حاصل از خوشه‌بندی را به ویژگی‌های اصلی اضافه کنید (و دوباره بهترین تعداد خوشه‌ها را جستجو کنید)، چه تغییری در عملکرد مدل ایجاد می‌شود.

۱۲. یک مدل مخلوط گاوسی (Gaussian Mixture Model) روی مجموعه‌داده‌ی چهره‌های Olivetti آموزش دهید. برای کاهش زمان محاسبات، بهتر است ابتدا ابعاد داده را کاهش دهید (برای مثال با PCA و حفظ حدود ۹۹% واریانس). سپس از مدل آموزش‌دیده برای تولید چهره‌های جدید با استفاده از متد ()sample استفاده کرده و آن‌ها را مصورسازی کنید. (اگر از PCA استفاده کرده‌اید، لازم است از متد ()inverse_transform آن بهره بگیرید تا تصاویر به فضای اصلی بازگردند.) در ادامه، چند تصویر را دستکاری کنید (برای نمونه، چرخاندن، وارونه‌کردن یا تیره‌تر کردن) و بررسی کنید که آیا مدل می‌تواند این تغییرات را به‌عنوان ناهنجاری تشخیص دهد یا خیر. برای این کار، خروجی متد ()score_samples را برای تصاویر معمولی و تصاویر تغییریافته مقایسه کنید.

۱۳. برخی روش‌های کاهش ابعاد را می‌توان برای تشخیص ناهنجاری نیز به‌کار برد. برای نمونه، مجموعه‌داده‌ی چهره‌های Olivetti را با استفاده از PCA کاهش دهید و حدود ۹۹% واریانس را حفظ کنید. سپس برای هر تصویر، خطای بازسازی را محاسبه کنید. در ادامه، تعدادی از تصاویر تغییریافته‌ای را که در تمرین قبل ساخته‌اید بررسی کنید و خطای بازسازی آن‌ها را اندازه بگیرید؛ خواهید دید که این خطا بسیار بزرگ‌تر است. اگر یکی از این تصاویر بازسازی‌شده را رسم کنید، دلیلش روشن می‌شود: مدل تلاش می‌کند آن تصویر تغییریافته را مانند یک چهره عادی بازسازی کند.

پاسخ‌های مربوط به این تمرین‌ها را می‌توانید در انتهای نوت‌بوک همین فصل، از طریق نشانی *https://homl.info/colab3* مشاهده کنید.